Minyi Guo Laurence T. Yang
Beniamino Di Martino Hans P. Zima
Jack Dongarra Feilong Tang (Eds.)

Parallel and Distributed Processing and Applications

LNCS 4330

4th International Symposium, ISPA 2006
Sorrento, Italy, December 2006
Proceedings

Springer

Lecture Notes in Computer Science 4330

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

Editorial Board

David Hutchison
Lancaster University, UK

Takeo Kanade
Carnegie Mellon University, Pittsburgh, PA, USA

Josef Kittler
University of Surrey, Guildford, UK

Jon M. Kleinberg
Cornell University, Ithaca, NY, USA

Friedemann Mattern
ETH Zurich, Switzerland

John C. Mitchell
Stanford University, CA, USA

Moni Naor
Weizmann Institute of Science, Rehovot, Israel

Oscar Nierstrasz
University of Bern, Switzerland

C. Pandu Rangan
Indian Institute of Technology, Madras, India

Bernhard Steffen
University of Dortmund, Germany

Madhu Sudan
Massachusetts Institute of Technology, MA, USA

Demetri Terzopoulos
University of California, Los Angeles, CA, USA

Doug Tygar
University of California, Berkeley, CA, USA

Moshe Y. Vardi
Rice University, Houston, TX, USA

Gerhard Weikum
Max-Planck Institute of Computer Science, Saarbruecken, Germany

Minyi Guo Laurence T. Yang
Beniamino Di Martino Hans P. Zima
Jack Dongarra Feilong Tang (Eds.)

Parallel and Distributed Processing and Applications

4th International Symposium, ISPA 2006
Sorrento, Italy, December 4-6, 2006
Proceedings

Springer

Volume Editors

Minyi Guo
The University of Aizu, Fukushima, Japan
E-mail: minyi@u-aizu.ac.jp

Laurence T. Yang
St. Francis Xavier University, Antigonish, NS, Canada
E-mail: lyang@stfx.ca

Beniamino Di Martino
Seconda Universitá di Napoli, Italy
E-mail: beniamino.dimartino@unina.it

Hans P. Zima
University of Vienna, Vienna, Austria/JPL, Caltech, USA
E-mail: zima@par.univie.ac.at

Jack Dongarra
University of Tennessee, Knoxville, TN, USA
E-mail: dongarra@cs.utk.edu

Feilong Tang
Shanghai Jiao Tong University, Shanghai, China
E-mail: tang-fl@cs.sjtu.edu.cn

Library of Congress Control Number: 2006938036

CR Subject Classification (1998): F.1, F.2, D.1, D.2, D.4, C.2, C.4, H.4, K.6

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

ISSN 0302-9743
ISBN-10 3-540-68067-5 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-68067-3 Springer Berlin Heidelberg New York

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

Springer is a part of Springer Science+Business Media

springer.com

© Springer-Verlag Berlin Heidelberg 2006
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11946441 06/3142 5 4 3 2 1 0

Preface

Welcome to the proceedings of the 4th International Symposium on Parallel and Distributed Processing and Applications (ISPA 2006), which was held in Sorrento, Italy, December, 4–6 2006.

Parallel computing has become a mainstream research area in computer science and the ISPA conference has become one of the premier forums for the presentation of new and exciting research on all aspects of parallel and distributed computing. We are pleased to present the proceedings for ISPA 2006, which comprises a collection of excellent technical papers and keynote speeches. The accepted papers cover a wide range of exciting topics including architectures, languages, algorithms, software, networking and applications.

The conference continues to grow and this year a record total of 277 manuscripts were submitted for consideration by the Program Committee. From these submissions the Program Committee selected only 79 regular papers in the program, which reflects the acceptance rate as 28%. An additional 10 workshops complemented the outstanding paper sessions.

The submission and review process worked as follows. Each submission was assigned to at least three Program Committee members for review. Each Program Committee member prepared a single review for each assigned paper or assigned a paper to an outside reviewer for review. In addition, the Program Chairs and Program Vice-Chairs read the papers when a conflicting review result occurred. Finally, after much discussion among the Program Chairs and Program Vice-Chairs, based on the review scores, the Program Chairs made the final decision. Given the large number of submissions, each Program Committee member was assigned roughly 7–12 papers.

The excellent program required a lot of effort from many people. First, we would like to thank all the authors for their hard work in preparing submissions to the conference. We deeply appreciate the effort and contributions of the Program Committee members who worked very hard to select the very best submissions and to put together an exciting program. We are also very grateful to the keynote speakers for accepting our invitation to present keynote talks. Thanks go to the Workshop Chairs for organizing ten excellent workshops on several important topics related to parallel and distributed computing and applications.

We deeply appreciate the tremendous efforts of the Program Vice-Chairs, Daniel S. Katz, Koji Nakano, Omer F. Rana, and Barbara Chapman. We would like to thank the conference Workshop Chairs, Geyong Min and Gudula Rünger, for their continued organization and support for all workshops.

We hope that attendees enjoyed this conference, found the technical program to be exciting, and had a wonderful time in Sorrento and its peninsula, together with the social activities of the conference.

Minyi Guo, Hans Zima
ISPA 2006 Program Co-chairs

Beniamino Di Martino, Jack Dongarra, and Laurence T. Yang
ISPA 2006 General Co-chairs

Organization

ISPA 2006 was organized by the Second University of Naples, Department of Information Engineering and Faculty of Political Studies, in collaboration with St. Francis Xavier University, University of Aizu, University of Tennessee, JPL-Caltech, and CREATE.

Executive Committee

General Chairs	Beniamino Di Martino, Second University of Naples, Italy
	Jack Dongarra, University of Tennessee, USA
	Laurence T. Yang, St. Francis Xavier University, Canada
Program Chairs	Minyi Guo, University of Aizu, Japan
	Hans Zima, University of Vienna, Austria/JPL, Caltech, USA
Program Vice-Chairs	Barbara Chapman, University of Houston, USA
	Koji Nakano, Hiroshima University, Japan
	Omer F. Rana, Cardiff University, UK
	Daniel S. Katz, LSU/JPL, USA
Workshop Chairs	Gudula Rünger, Chemnitz University of Technology, Germany
	Geyong Min, University of Bradford, UK
Publicity Chairs	Tomoya Enokido, Rissho University, Japan
	Xubin (ben) He, Tennessee Technological University, USA
	Man Lin, St. Francis Xavier University, Canada
Publication Chairs	Feilong Tang, Shanghai Jiao Tong University, China
	Geyong Min, University of Bradford, UK
Organizing Committee	Beniamino Di Martino (Chair), Salvatore Venticinque Francesco Moscato, Domenico Di Sivo, Valentina Casola, Patrizia Petrillo, Second University of Naples, Italy
	Man Lin, Tony Li Xu, St. Francis Xavier University, Canada

Program Committee

"Languages and Algorithms" Track

Ulrich Ruede	University of Erlangen,Germany
Marc Garbey	University of Houston, USA

Program Committee (continued)

Michael Philippsen	University of Erlangen, Germany
Iain Duff	Rutherford Labs, UK
Henk Sips	TU Delft, Netherlands
Lois Curfman McInnes	Argonne National Lab, USA
Yuri V. Vassilevski	Academy of Sciences, Russia
Wei Shyy	University of Michigan, USA
Luc Giraud	ENSEEIHT, France
Frederic Desprez	INRIA, France
Jingling Xue	University of New South Wales, Australia
Tsung-Chuan Huang	National Sun Yet-sen University, Taiwan
Michela Taufer	University of Texas at El Paso, USA
Baoliu Ye	University of Aizu, Japan
Young-Sik Jeong	Wonkwang University, Korea
Antonino Mazzeo	University "Federico II" of Naples, Italy
Nicola Mazzocca	University "Federico II" of Naples, Italy

"Architectures and Networks" Track

Jacir L. Bordim	Brasilia University, Brazil
Anu Bourgeois	Georgia State University, USA
Satoshi Fujita	Hiroshima University, Japan
Akihiro Fujiwara	Kyushu Institute of Technology, Japan
Shuichi Ichikawa	Toyohashi University of Technology, Japan
Yasushi Inoguchi	JAIST, Japan
Chuzo Iwamoto	Hiroshima University, Japan
Xiaohong Jiang	Tohoku University, Japan
Hirotsugu Kakugawa	Osaka University, Japan
Hiroshi Matsuo	Nagoya Institute of Technology, Japan
Susumu Matsumae	Tottori University of Environmental Studies, Japan
Eiji Miyano	Kyushu Institute of Technology, Japan
Dajin Wang	MontClair State University, USA
Jose A. F. Zepeda	CICESE, Mexico
Jingyuan Zhang	University of Alabama, USA
Umberto Villano	University of Sannio, Italy
Rocco Aversa	Second University of Naples, Italy
Massimiliano Rak	Second University of Naples, Italy
Salvatore Venticinque	Second University of Naples, Italy

"Middleware and Cooperative Computing" Track

Noria Foukia	University of Otago, New Zealand
Simon Miles	University of Southampton, UK
Annika Hinze	University of Waikato, New Zealand
Suleiman Hassein	University of Cape Town, South Africa
Jonathan Giddy	Welsh eScience Centre, UK

Program Committee (continued)

Elth Ogston	Free University Amsterdam, Netherlands
Laslo Varga	SZTAKI, Hungary
Syed Naqvi	ENST, France
Benno Overeinder	Free University, Amsterdam, Netherlands
Hafiz Farooq	COMTEC, Japan
Daniel Olmedilla	University of Hannover, Germany
Roy Sterritt	University of Ulster, UK
A.E.F. Seghrouchni	University of Paris 6, France
Biplab Kumer Sarker	University of New Brunswick, Canada
Chao-Tung Yang	Tunghai University, Taiwan
Kuan-Ching Li	Providence University, Taiwan
Feilong Tang	Shanghai Jiao Tong University, China
Francesco Moscato	Second University of Naples, Italy
Valentina Casola	University "Federico II" of Naples, Italy
Stefano Marrone	Second University of Naples, Italy

"Software and Applications" Track

Mark Baker	University of Portsmouth, UK
Patrick Bridges	University of New Mexico, USA
Rajkummar Buyya	University of Melbourne, Australia
Wentong Cai	Nanyang Technological University, Singapore
Guojing Cong	IBM T.J. Watson Research Center, USA
Ewa Deelman	USC Information Sciences Institute, USA
Wolfgang Gentzsch	D-Grid/RENCI, Germany/USA
Kazuki Joe	Nara Women's University, Japan
Hiroaki Kobayashi	Tohoku University, Japan
Christine Morin	IRISA/INRIA, France
Marcin Paprzycki	Computer Science Institute, SWPS, Poland
Heinz Stockinger	University of Vienna, Austria
Alan Sussman	University of Maryland, USA
Cho-Li Wang	The University of Hong Kong, Hong Kong
Shujia Zhou	NASA Goddard Space Flight Center, USA
Yiming Li	National Chiao Tung University, Taiwan
Rodrigo de Mello	University of São Paulo, Brazil
Franco Frattolillo	University of Sannio, Italy
Kuan-Ching Li	Providence University, Taiwan

Additional Reviewers

Marco A. S. Netto	Shunsuke Araki	Eunjung Cho
Nael Abuhalaweh	Abhinav Bhatele	Alessandro Cilardo
Georges Amvame Nze	Eddy Caron	Jos M. Claver

Luigi Coppolino
Giusy Di Lorenzo
Marcos D. de Assuncao
Joerg Diederich
Falko Dressler
Yaakoub El Khamra
Brett Estrade
Masaru Fukushi
Li Gao
Isabelle Guerin Lassous
Suman Gupta
Bernd Hardung
Masatsugu Hashimoto
Lei Huang
Nisar Hundewale
Andrei Hutanu
Alexandru Iosup
Fakhra Jabeen
Roger Karrer
Shigeru Kashihara
Kenji Kawahara
Andre Kerstens
Jik-Soo Kim
Kyong Kim
Moonseong Kim
Keiji Kimura
Kenji Kise
Gilles Klein
Arnaud Legrand
Kamesh Madduri
Akihiro Musa
Krishna Nadiminti
Beomseok Nam
Mariusz Nowostawski
Jaesung Park
Laura Pearlman
Eric Platon
Shoichi Saito
Hiroyuki Sato
Garry Smith
Eiko Sugawara
Masami Takata
Hiroyuki Takizawa
Kiyofumi Tanaka
Tomoaki Tsumura
Hitoshi Uda
Zhijun Wang
Joe Shang-Chieh Wu
George Xylomenos
Jaepil Yoo
Il-Chul Yoon
Liu Yun
Saber Zrelli

Table of Contents

Session 4. Algorithms

Track 3. Middleware and Cooperative Computing

Session 5. Middleware

Session 6. Cooperative Computing

The Walls of Computer Design

Jean-Luc Gaudiot

Department of Electrical Engineering and Computer Science,
University of California
gaudiot@uci.edu
http://pascal.eng.uci.edu/people/gaudiot.html

Most of the work of the computer architect today has to do with bridging the well-known gigantic gap between processor speed and memory access time. However, other significant challenges are looming on the horizon. For one thing, higher device density and smaller design rules are increasing the sensitivity of circuits to outside radiation events. Also, power issues are creating significant challenges, not only in terms of power consumption for embedded devices, but also in terms of cooling and power dissipation. Hence, while Moore's law will continue to hold for the foreseeable future, processor speeds are not expected to follow suit, thereby requiring new architectural concepts.

In this lecture, we will describe the issues surrounding modern computer architecture and design. We will discuss the advent of newer architectures such as multi-core chips and examine the associated synchronization problems, implementation issues, and performance evaluation considerations. We will also study the implications of power considerations and demonstrate scheduling approaches to control consumption and heat dissipation. Redundant thread execution will be demonstrated as a simple synchronization method which offers low cost protection against the Single Event Upsets (SEUs) plaguing modern systems with increasing frequency.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, p. 1, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The Impact of Multicore on Math Software and Exploiting Single Precision Computing to Obtain Double Precision Results

Jack Dongarra

Computer Science Department
University of Tennessee and Oak Ridge National Laboratory
dongarra@cs.utk.edu
http://www.cs.utk.edu/ dongarra/

Recent versions of microprocessors exhibit performance characteristics for 32 bit floating point arithmetic (single precision) that is substantially higher than 64 bit floating point arithmetic (double precision). Examples include the Intel Pentium IV and M processors, AMD Opteron architectures, the IBM Cell processor and various GPUs. When working in single precision, floating point operations can be performed up to two times faster on the Pentium and up to ten times faster on the Cell over double precision. The motivation for this work is to exploit single precision operations whenever possible and resort to double precision at critical stages while attempting to provide the full double precision results. The results described here are fairly general and can be applied to various problems in linear algebra such as solving large sparse systems, using direct or iterative methods and some eigenvalue problems. There are limitations to the success of this process, such as when the conditioning of the problem exceeds the reciprocal of the accuracy of the single precision computations. In that case the double precision algorithm should be used.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, p. 2, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Implementing Tomorrow's Programming Languages

Rudolf Eigenmann

School of Electrical and Computer Engineering
Purdue University 465 Northwestern Ave., West Lafayette, IN 47907, USA
eigenman@purdue.edu

Compilers are the critical translators that convert a human-readable program into the code understood by the machine. While this transformation is already sophisticated today, tomorrow's compilers face a tremendous challenge. There is a demand to provide languages that are much higher level than today's C, Fortran, or Java. On the other hand, tomorrow's machines are more complex than today; they involve multiple cores and may span the planet via compute Grids. How can we expect compilers to provide efficient implementations? I will describe a number of related research efforts that try to tackle this problem. Composition builds a way towards higher-level programming languages. Automatic translation of shared-address-space models to distributed-memory architectures may lead to higher productivity than current message passing paradigms. Advanced symbolic analysis techniques equips compilers with capabilities to reason about programs in abstract terms. Last but not least, through auto-tuning, compilers make effective decisions, even through there may be insufficient information at compile time.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, p. 3, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Trends in High Performance Computing for Industry and Research

Frank Baetke

HPC - Technology Program Manager Global
frank.baetke@hp.com

Today trends in High Performance Computing can be separated into two basic categories: one covers the directions towards smaller, faster and hopefully cooler components, the other includes more innovative directions, which might lead to real paradigm shifts.

Of course, it became obvious in the past years that the HPC scene is and will be dominated by all kinds of clusters. The talk will address the various aspects we are seeing today in terms of processors, node architectures and operating systems. The concept of heterogeneous clusters as well as the challenge of cluster file-systems and integrated visualization will be covered shortly.

The talk will shortly cover new trends in cluster management and specific aspect in industrial and research environments. One strategic approach comprises a triangle with computation, data management and visualization representing the sides of the triangle. The successes of a standard-based strategy will be contrasted with more dedicated and specialized approaches.

Since last year, certain changes became obvious specifically in the area of components, which make up the computational parts. Those changes provide vast opportunities in terms of system efficiency but will also lead to new challenges due to the vastly increased densities of advanced architectures.

For all the topics mentioned above characteristic architectures and sites will be shown as examples. A few hints concerning roadmaps and future issues will be added.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, p. 4, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Emerging Technologies and Large Scale Facilities in HPC

Marco Briscolini

HPC & Deep Computing Sales Consultant EMEA Southwest
IBM Italia S.p.A.
marco briscolini@it.ibm.com
http://www.ibm.com/servers/hpc

Recently developments in microprocessor technology, high performance interconnection, storage subsystem, and grid infrastructure, address pflops and pbytes capabilities for large scale architectures which will categorize large HPC facilities in the near future.

IBM will attack such new challenges with a broad range of collaborations in open technologies with major technological partners and integrating and clustering more and more efficiently and reliably most of innovative solutions. The basic idea is to combine innovation and outstanding performances at competitive costs.

In such scenario talk will cover an overview of IBM solutions and future trends for HPC with an insight on integrated solutions as Blue Gene and Cell Broadband Engine (CBE) processor, this latter a product recently announced based on Power Architecture and originally developed in collaboration with IBM, Sony Corporation, Sony Computer Entertainment Inc, and Toshiba Corporation.

Some considerations on high performance storage solutions as those based on GPFS and SFS/SVC will conclude as an example of designing I/O infrastructures locally or geographically distributed and able to provide a balance from high density computing power and large I/O capabilities.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, p. 5, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Architecture and Performance of Dynamic Offloader for Cluster Network

Keiichi Aoki, Hiroki Maruoka, Koichi Wada, and Masaaki Ono

Department of Computer Science, University of Tsukuba
1-1-1 Tennodai, Tsukuba, Ibaraki, Japan
k1@padc.cs.tsukuba.ac.jp, wada@cs.tsukuba.ac.jp

Abstract. This paper presents an architecture of the dynamic offloading mechanism, called Maestro Dynamic Offloading Mechanism(MDO), for the intelligent cluster network Maestro2. By using MDO, programmers can offload software modules to the network interface and the switch dynamically. MDO provides programmers functional APIs with which programmers can develop offload modules efficiently. MDO also includes wrapper library that enables the offload modules to be executed on the host processors as well as on the network devices. The results of performance evaluation showed that the performance of the collective communications can be improved by offloading communication procedures to the network devices using MDO. The overhead of the MDO and the traffic on PCI bus are also discussed.

1 Introduction

With the advent of high performance networks for clusters, such as Myrinet[1], QsNet[2], and InfiniBand[3], many works have been done on developing low latency and high throughput communication software. GM[4] and PM[5] are the examples of those communication libraries, which improve the delivered communication performance by introducing zero-copy communication. On the other hand, such communication libraries consume a sizable amount of computing power of the host processor to process communication protocol and to control network hardware[6][7]. This disturbs the host processor to perform calculation of an application and results in the performance degradation of a cluster. Offloading communication process to network devices is a promising technique to remedy this issue, since it can make the host processor concentrate on its main computation and improve the overall efficiency of clusters.

We can find several researches or products for offloading communication protocol to network devices such as TOE[8] or offloaded-MPI[9]. Although they have succeeded in increasing communication performance, the host processor still has to control network hardware to handle communication frequently because the unit of offloading is small. By offloading whole communication library or a part of the application, the host processor can be released from the burden of communication procedure. This increases an opportunity of overlapping computation with communication.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 6–17, 2006.
© Springer-Verlag Berlin Heidelberg 2006

To offload large software modules that are used to be processed on a host processor to network devices, network devices are required to introduce a high performance processor instead of special purpose processor such as a network processor, and high-capacity memory. We have developed a high performance cluster network called Maestro2[10]. Maestro2 is composed of network interfaces(NI) and switch boxes(SB). Both an NI and an SB include a general purpose processor and a high-capacity memory so that user-defined software modules can be performed.

We have also developed a message passing library that can extract maximal performance of the network, named MMP[11]. In MMP, the firmware of MMP that resides on NI controls network hardware and handles protocol processing such as an addition of message header. The interface functions of MMP are designed as non-blocking to overlap computation in application programs and communication processing. In MMP, communication is done by issuing a send/receive request to NI with few parameters. Then, the NI performs communication independently from the host processor. Thus, the offloading in MMP is more aggressive than protocol offloading. We confirmed that this design freed host processor and host bus from communication processing and increased throughput of communication. Through the development of MMP, we have found that communication performance has been effectively increased by offloading communication processing to network devices.

In this paper, a dynamic offloading mechanism for Maestro2 cluster network, which can offload user-defined software modules dynamically to the network devices, is proposed. This mechanism includes a wrapper library that enables the offload modules to be executed on the host processors as well as on the network devices.

2 Maestro2 Cluster Network

As described in the previous section, we have developed Maestro2 cluster network which has the capability of executing user-defined software. In this section, we will describe briefly the architecture and implementation of Maestro2 cluster network.

Maestro2 cluster network is comprised of network interfaces and switch box as shown in Fig. 1. Network interfaces are connected to each host processor via the PCI bus and exchange message with host processor. A message is a communication unit composed of one or more packets. The switch box is connected to up to eight network interfaces via LVDS (Low Voltage Differential Signaling) physical layer[12] and is responsible for switching messages.

(1) Network interface
The network interface (NI) is composed of a LVDS transmitter/receiver, 8K-bytes network buffer, MLC-X link layer protocol, PCI interface, PowerPC603e[13] 300MHz, 64M-bytes SDRAM, MLC-X, PCI interface, and network buffer are implemented in Xilinx VirtexII FPGA chip[14]. MLC-X controls LVDS transmitter/receiver bidirectionally. LVDS transmitter/receiver is

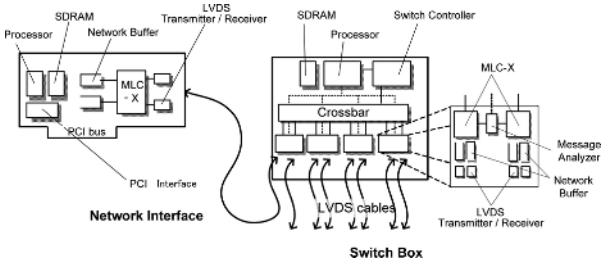


Fig. 1. Architecture of Maestro2

connected to the switch box and transfers data at 6.4Gbps (3.2Gbps + 3.2Gbps full-duplex). The PCI interface maps the address space of the SDRAM into the host processor's address space and host processor's memory space into the PowerPC's address space.

(2) Switch box
The switch box (SB) currently includes eight LVDS transmitter/receiver, four SB interfaces, a crossbar switch, PowerPC603e 300MHz, 32M-bytes SDRAM, and switch controller. SB interfaces are composed of a message analyzer, two MLC-X link layer protocol, and 16K-bytes network buffer. It communicates with network interfaces via LVDS connections. MLC-Xs and network buffers in the switch box are similar to the ones in the network interface. Each message analyzer is connected to two MLC-Xs. It picks up headers from messages and passes them to the PowerPC processor. PowerPC analyses headers received from the message analyzer and controls the switch controller.

As described above, both of the network interface and the switch box of Maestro2 cluster network has powerful processor and large-capacity RAM on boards. By using this processor, Maestro2 cluster network can control hardware and handle protocol or other communication processing independently of host processors. Thus the computation of the application program on the host processor and the communication can be overlapped significantly.

3 Maestro Dynamic Offloading Mechanism

3.1 Architecture of MDO

Maestro dynamic offloading mechanism (MDO) is the software environment to offload user-defined modules to Maestro2 cluster network. MDO is composed of one or more user-defined modules, the MDO library, the module wrapper library, and the firmware for the network interface and the switch box. Figure 2 summarizes the architecture of MDO.

The firmware is currently implemented as part of the MMP firmware for the network interface and the switch box. It loads the modules transferred from the

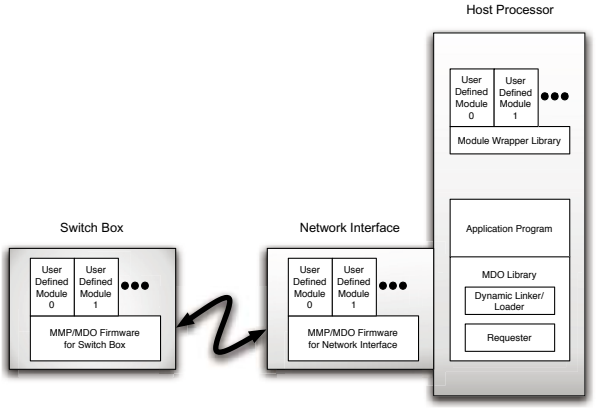


Fig. 2. Architecture of MDO

MDO library. Additionally it calls the modules when it receives a request for the module from the MDO library or receives the message for the module from network.

The MDO library is the library for application programs on a host processor. It includes dynamic linker/loader to load user-defined modules dynamically into network devices and requester to handle requests for modules from application programs. User's application program can load necessary modules to network devices and issue requests to the module by using MDO library.

User-defined modules include native executable binary for the network interface or the switch box and the information for relocation. When they are called from the firmware, they handle requests from the application program via the MDO library and control peripheral network hardware. Besides, by being recompiled, the user-defined modules for the network interface can be executed on the host processor without any changes in its source codes. This is useful for debugging and examining adequate load partitioning between the host and the network interface. When they are compiled as modules for host processors, the MDO library calls them via the module wrapper library. The MDO library also hooks function calls from user-defined modules for controlling network hardware by translating hardware dependent value, such as physical address, to make the module to work correctly on a host processor.

3.2 Interface Functions of MDO

The MDO library provides interfaces to application programs for loading modules and issuing requests to them. We will detail about these interfaces below.

(1) `MMP_Regist_mod_NI(...)`
This function relocates the specified module and loads it to unused memory region on the network interface. To relocate modules, this function investigates address informations of loaded firmware and builds the relocation

table. Then it resolves symbols' addresses in the module with referring the relocation table. Finally, it transfers the relocated module and registers the information of module such as functions' entry addresses to the firmware on the network interface.

(2) `MMP_Regist_mod_SB(...)`
This function registers the module to the switch box. It relocates the module in the same method as `MMP_Regist_mod_NI()`. Then it transmits the relocated module to the switch box through the network interface.

(3) `MMP_Mod_req(...)`
When the application program needs to request the module on the network interface to process something, this function is able to pass the request and the arguments to the module on network interface. The arguments passed from the application program are stored to request queue on the network interface.

(4) `MMP_Mod_recv_post(...)`
If the module has possibilities to transfer messages from network to the memory on host processor, the module must be able to set the destination address of the host processor to DMA hardware. The application program could pass the address information through this function beforehand. Because the module is able to calculate address, the passed address might be only the base address or the offset.

(5) `MMP_Get_mod_req_stat(...)`
This function enables an application program to receive calculation results from the module or wait for completion of communication processing in the module.

Meanwhile, the MDO module has to contain callback functions to handle requests from the application program and messages from other modules. Details of these functions are as follows:

(1) `NI_mod_hostreq_handler(...)`
Corresponding to the requests issued from an application program by `MMP_Mod_req()`, this function in the module loaded on network interface is called.

(2) `NI_mod_recvmsg_handler(...)`
When the firmware on network interface receives a message, it calls this function in the module. Generic information such as the size of the message and the source address of the message is passed as arguments. Pre-post information for reception from application program by using `MMP_Mod_recv_post()` is also passed if it exists.

(3) `SB_mod_recvheader_handler(...)`
The firmware on switch box calls this function whenever it receives the message to the module on switch box from network interfaces. The module controls the hardware on switch box to transfer the message to other network interface(s) or to the memory on the switch box to process data included in the message.

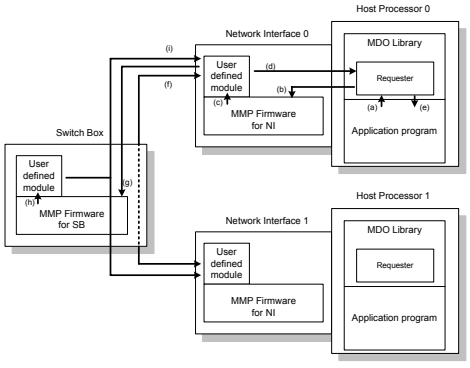


Fig. 3. Inter-process communication in MDO

3.3 Inter-process Communication in MDO

In MDO, inter-process communication can be categorized by two types: a) request exchange between the application program on host processor and the module on network interface. b)message exchange between modules on network interfaces or switch boxes. Figure 3 depicts communication details when a module is loaded on network interfaces and the switch box.

The application program on host processor can issue requests with arguments through MDO library to the module on network interface(Fig.3 (a),(b)). When the application program issues the requests to the module, the firmware on network interface calls the function with specific symbol name in the module with arguments from the application program(Fig.3 (c)). The module may transmit several messages per single request if necessary, or it may transmit nothing and do only computation. If the module needs to return the result to the application program, the result can be returned via the MDO library(Fig.3 (d),(e)).

Modules on network interfaces and on switch boxes can exchange messages each other without requests from host processors. Modules on network interface can exchange messages each other through the switch box (Fig.3 (f)). Thus, with MDO, host processor can be freed from communication burden such as relaying messages performed in collective communication. This reduces latency in host processor's bus or overheads in frequent interaction between communication library and an application program, and increases communication performance.

The module on the network interface and the one on the switch box can also exchange messages in MDO. When the firmware on switch box receives messages from the module on network interface, it calls the module on switch box(Fig.3 (g)(h)). The callee module can transmit messages to one or more modules on the network interface after executing designated procedures(Fig.3 (i)). This feature is effective to reduce the number of messages in network and the number of phases of communication when a module or an application program needs to broadcast, or sum up the calculation results in multiple hosts.

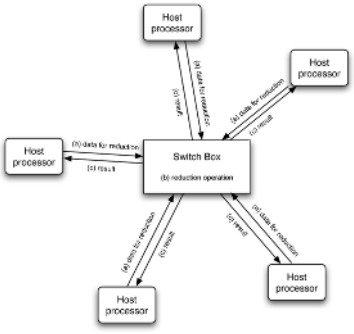


Fig. 4. Collective communication with MDO

3.4 Offloading Collective Communication

We have implemented three collective communications by using MDO in this research: synchronization, broadcast, and allreduce.

When a host processor transmits messages to several other hosts in these communications, the module on switch box communicates with host processors. Figure 4 shows communication flow in the collective communication with MDO. When the application program needs reduction operation during calculation, host processors send their own data to the module on switch box. Upon receiving the data, the module performs reduction operation. Finally the module transmits results to all the hosts simultaneously. Therefore, the number for message needed in the collective communication is decreased compared with the peer-to-peer communication between host processors. Moreover, the number of steps in communication becomes constant even if the number of host processors increases.

Computations that are necessary for allreduce could be overlapped with computations of the application program on a host processor, because modules on network devices could compute independently of host processors. As a result, the total time for application program can be reduced.

4 Evaluation

4.1 Experimental Environment

We prepared the environment as shown in Table 1 for experiments. We performed three experiments on this environment. We use `gettimeofday()` function to measure the time on the host processor in all experiments.

4.2 Overheads and Advantages of MDO

The first experiment is performed to evaluate the overheads of MDO. Three programs are executed that synchronize all processes on multiple hosts. The

Table 1. Experimental environment

CPU	Intel Xeon 2.8GHz
Memory	1G-Bytes
Network	Maestro2 cluster network
OS	Debian GNU/Linux 3.1

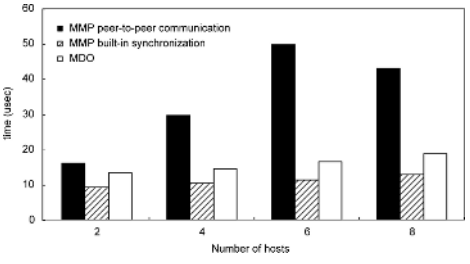


Fig. 5. Time of synchronization using MMP built-in function, MDO, and MMP peer-to-peer function

first one uses MMP's built-in function to synchronize processes. The second one uses the module for synchronization and MDO. The third one uses MMP's send/receive functions and the tree-based synchronization algorithm with peer-to-peer communication used in MPICH[15].

The results are shown in Fig. 5. The MMP's built-in function used in the first program is implemented in monolithic firmware and library of MMP. On the other hand, synchronization requests in the second program are processed by dynamically loaded module of MDO. Therefore, the difference between the results of the first and the second programs shows the overhead of calling module in MDO. This result shows that the overhead is about 30% of whole synchronization time.

Then, we compared the result of MDO with the result of peer-to-peer communication. The results show that the synchronization program using MDO is up to three times faster than the one using peer-to-peer communication.

4.3 Performance of Broadcast

In this section, we measured the time to complete broadcasting with varying data size and the number of hosts. Broadcast has been performed by MMP peer-to-peer communication and MDO. Figure 6 shows the time using eight processors with varying data size. Figure 7 shows the time for broadcasting 64K-bytes message with varying the number of hosts.

Both results show that the broadcast using MDO is always faster than the one using peer-to-peer communication between host processor. In Fig. 6, we found MDO requires only 38% of the time of peer-to-peer at the minimum. In

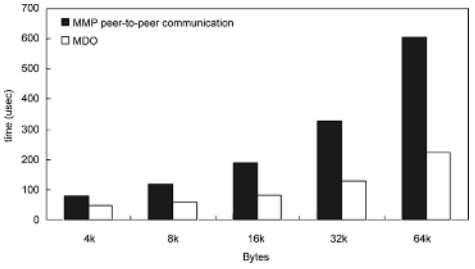


Fig. 6. Time of broadcast using MDO and MMP peer-to-peer function with varying data size

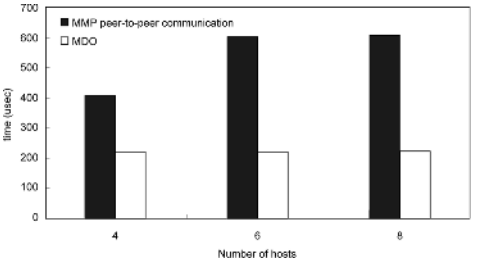


Fig. 7. Time of broadcast using MDO and MMP peer-to-peer function with varying the number of hosts

Fig. 7, the time of MDO is almost constant even when the number of hosts increases. And the time of MDO is up to 63% smaller than the one of peer-to-peer communication. From these results, we confirmed that MDO can effectively reduce time for broadcast.

4.4 Performance of Allreduce

Finally, we measured the performance of allreduce with varying the number of host processors. In this experiment, we developed the module which calculates the product of values received from host processors and returns the result to all processors. We also developed equivalent function by using MMP's peer-to-peer communication as is the case in the experiment of broadcast, and we compared the result of MDO with the result of peer-to-peer communication.

Figure 8 shows the results of this experiment. As this figure shows, MDO was able to reduce the time for communication up to 39%.

4.5 Comparison of Traffic over PCI Bus

We compared the amounts of data transferred via host processors' PCI bus when they perform broadcast and allreduce with MDO and MMP peer-to-peer communication. In Fig. 9, we can see the traffic over PCI bus when four and eight hosts perform broadcast and allreduce. The traffic consists of data and

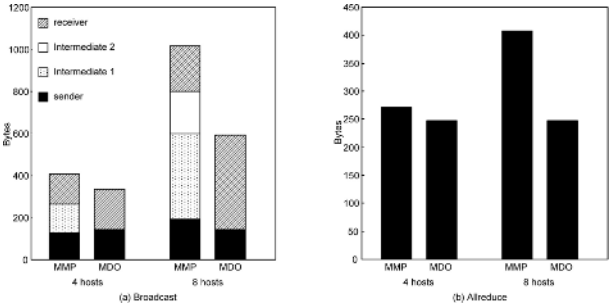


Fig. 8. Time of allreduce using MDO and MMP peer-to-peer function

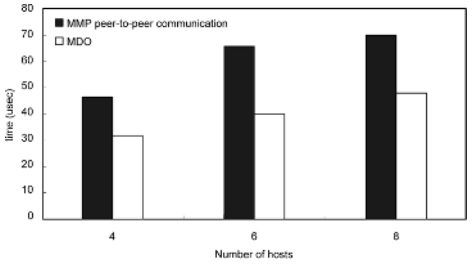


Fig. 9. Comparison of traffic over PCI

control commands for the network interface. All the traffic occurred on PCI bus of all hosts are accumulated and shown in Fig. 9.

Broadcast based on the peer-to-peer communication requires intermediate hosts, which re-send the message soon after they receive, for relaying. This relaying yields traffic on PCI bus. The intermediate 1 and 2 in Fig. 9 (a) indicate the first and the second intermediate hosts. In broadcast with four hosts, one host works as intermediate 1. In broadcast with eight hosts, three hosts work as intermediate 1 and one host works as intermediate 2. On the other hand, broadcast by MDO requires no intermediate hosts because the switch box generates and sends the copies of the message for broadcast. As a result, traffic on PCI bus is reduced by 18% and 42% of peer-to-peer communication in the case of four and eight hosts, respectively.

Meanwhile, all hosts must send and receive data several times in allreduce by peer-to-peer communication. The number of send and receive operations is in the order of $\log n$, where n is the number of hosts. By contrast, all reduce by MDO requires only one send and receive operations in each host as described in the Sect. 3.4. Figure 9 (b) shows that the reduction ratios of the traffic on PCI are 9% and 39% for four hosts and eight hosts, respectively. It has been confirmed that MDO reduces more traffic on PCI when the number of hosts involved in allreduce increases.

From the results shown in Fig. 9, MDO can reduce the traffic on PCI bus for both broadcast and allreduce operations. The reduction of the traffic improves

efficiency of PCI bus and contributes to reduce execution time of collective communication.

5 Conclusion

In this paper, we presented the software environment that allows the application program to offload user-defined modules to Maestro2 cluster network, named Maestro dynamic offloading mechanism (MDO). MDO includes user-defined modules, MDO library to offload user-defined modules to the network interface and/or switch box dynamically, and the firmware to invoke the loaded modules. Moreover, we described the implementation of broadcast and allreduce by using MDO and showed results of evaluation.

In the evaluation to see the performance impacts of MDO, we performed three experiments: synchronization, broadcast, and allreduce. The evaluation results showed that MDO is effective in reducing time of those collective communication, even though MDO requires extra time for offloading and invoking modules. Additionally, we measured the traffic over PCI bus when the hosts perform broadcast and allreduce with and without MDO. From the results, we have confirmed that MDO can reduce the traffic on PCI bus by approximately 40% compared to peer-to-peer communication.

For the future works, we will evaluate the performance of MDO by offloading a part of an application. We are also planning to develop distributed shared memory system using MDO mechanism. Furthermore, we are currently developing next generation of Maestro2 cluster network which has more powerful processor and higher-capacity memory on board. We will evaluate the performance impacts of offloading user-defined module by using this next-generation network.

Acknowledgement. This research was supported by Japan Society for the Promotion of Science, a Grant-in-Aid for Scientific Research(B), No. 16300012.

References

1. Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.K.: Myrinet – a gigabit-per-second local-area network. IEEE Micro **Vol.15, No.1** (1995) 29–35
2. Beecroft, J., Addison, D., Petrini, F., McLaren, M.: QsnetII: An interconnect for supercomputing applications. Technical report, Quadrics Ltd. (2003)
3. Infiniband Trade Association: InfiniBand Architecture Specification, Release1.0. (2000)
4. Myricom, Inc: The GM Message Passing System. (1999)
5. Takahashi, T., Sumimoto, S., Hori, A., Harada, H., Ishikawa, Y.: PM2: High performance communication middleware for heterogeneous network environment. In: proceedings of the IEEE/ACM SC2000 Conference. (2000) 52–53
6. Pfister, G.F.: An introduction to the infiniband architecture. In Hai, J., Toni, C., Buyya, R., eds.: High Performance Mass Storage and Parallel I/O. John Wiley & Sons Inc (2001) 617–632

7. Bierbaum, N.: MPI and embedded TCP/IP gigabit ethernet cluster computing. In: Proceedings of the 27th Annual IEEE Conference on Local Computer Networks (LCN'02). (2002) 733
8. Mogul, J.C.: TCP offload is a dumb idea whose time has come. In: Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX). (2003) 25–30
9. Brightwell, R., Underwood, K.D.: An analysis of NIC resource usage for offloading MPI. In: Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04). (2004)
10. Aoki, K., Yamagiwa, S., Ferreira, K., Campos, L.M., Ono, M., Wada, K., Sousa, L.: Maestro2: High speed network technology for high performance computing. In: Proceedings of 2004 IEEE International Conference on Communication (ICC2004). Volume 2. (2004) 1033–1037
11. Aoki, K., Wada, K., Ono, M., Yamagiwa, S.: High performance message passing library for maestro2 cluster network. In: Proceedings of the 23rd IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN 2005). (2005) 199–204
12. IEEE Standard Department: IEEE Standard for Low-Voltage Differential Signals (LVDS) for Scalable Coherent Interface (SCI). (1996)
13. Freescale Semiconductor, Inc: MPC603e RISC Microprocessor User's Manual. (2002)
14. Xilinx Inc: Virtex-II Platform FPGA Data Sheet. (2002) http://www.xilinx.com.
15. Gropp, W., Lusk, E., Doss, N., Skjellum, A.: A high-performance, portable implementation of the MPI message passing interface standard. Parallel Computing **22**(6) (1996) 789–828

Session Key Forwarding Scheme Based on AAA Architecture in Wireless Networks

Jong-Hyouk Lee and Tai-Myoung Chung

Internet Management Technology Laboratory,
Dept. of Computer Engineering, Sungkyunkwan University,
300 Cheoncheon-dong, Jangan-gu,
Suwon-si, Gyeonggi-do, 440-746, Korea
{jhlee, tmchung}@imtl.skku.ac.kr

Abstract. Due to an increasing number of portable devices, supports for quality of service (QoS) and security problems become significant issues in wireless networks. For this reason, Authentication, Authorization, and Accounting (AAA) architecture has been proposed. However AAA architecture has inefficient authentication procedures when a mobile node (MN) roams to a foreign network because AAA architecture assume that the only reliable source of authentication is an AAA server in the home network. In this paper, we propose session key forwarding scheme that can reduce an authentication time and an authentication failure rate while maintaining a security level. The performance results show that the proposed scheme reduces the authentication latency up to 6.16% and the authentication failure rate up to 23.9% compared to the standard AAA architecture.

1 Introduction

The tremendous advance of wireless communication technologies has facilitated the ubiquitous Internet service and also lots of popular applications including e-business require transmission of highly sensitive information often over wireless networks, while inducing more challenges to security due to open medium transmission [1]. In order to provide security services in wireless networks, authentication is used as an initial process to authorize a MN for communication through secret credentials such as AAA architecture [2,3]. Currently, AAA architecture for the MNs rely on frequently consulting an AAA home-server (AAAH) to authenticate the MNs when the MNs roams to a foreign network because an AAA local-server (AAAL) has no information about them. Although the AAAL may have a indirect trust relationship with the AAAH that stores the credentials for the MNs, the credentials of the MNs are encrypted and transmitted for remote verification hop-by-hop between the AAAL and AAAH. According to this reason, each time the MN visits a new foreign network, the AAAL has to send back an authentication request to the AAAH to be verified [4,5].

This problem becomes more significant when the MN is far away from the home network. The MN will be spoofed by attackers who send fake registration

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 18–26, 2006.
© Springer-Verlag Berlin Heidelberg 2006

requests to MN's home agent (HA) to redirect the messages to themselves or deny the service for the MN unless the messages be encrypted. Moveover, the foreign agent (FA) and AAAL do not generally have the information to authenticate the MN when the MN come to the foreign network for the first time. Also, the messages between the FA and HA should be encrypted and protected from attackers.

In order to resolve these problems, this paper proposes session key forwarding scheme between foreign networks. When a MN moves from a foreign network to another foreign network, the new AAAL (nAAAL) in new foreign network relies on the previous AAAL (pAAAL) to authenticate the MN. Therefore the MN is authenticated from the nAAAL without request to the AAAH. Consequently, this scheme reduces the authentication latency up to 6.16% and the authentication failure rate up to 23.9% compared to the standard AAA architecture.

This paper is organized as follows. In section 2, we introduce the concepts of AAA architecture and security association. Our proposed scheme is discussed in section 3. In section 4, we evaluate the performance of the proposed scheme. Finally we conclude the paper in section 5.

2 Preliminaries

The IETF AAA Working Group has been working for several years to establish a general architecture for Authentication, Authorization, and Accounting (AAA). It is shown in Fig. 1. When a MN is crossing the boundaries of different foreign network with an on-going service, an inter-domain handoff authentication occurs. Since the security association (SA) attached with the on-going communication session is between the MN and FA, no SA exists between the MN and FA, so it is necessary to contact the AAAH of the MN for authentication [6].

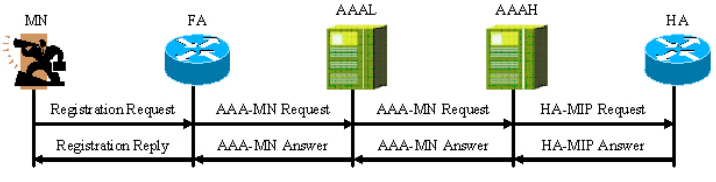


Fig. 1. AAA Architecture

In Fig. 1, when the AAAL receives the AAA-MN Request forwarded from the FA, the AAAL must forward the AAA-MN Request to the AAAH of the MN for verification because the AAAL does not have enough information to authenticate the visiting MN. After authentication at the AAAH, the AAAH sends the HA-MIP Request to the home agent (HA) and then the HA registers the MN. The HA creates the HA-MIP Answer including HA-MN and FA-MN keys, which are sent by the AAAH and then the AAAH sends AAA-MN Answer to the AAAL. Upon receiving the message, the AAAL forwards it to the FA.

The FA sends the Registration Reply created by the HA to the MN. Finally, the MN checks the Registration Reply and obtains the required keys (FA-MN key, HA-MN) to protect the communication.

As you see in Fig. 2(a), current AAA architecture causes long authentication delay because the AAAL must request the MN's authentication to the AAAH of the MN for verification. Each time the MN visits a new foreign network from a previous foreign network, a nAAAL has to request the MN's authentication to the AAAH even though a pAAAL has the session keys for HA-MN and pFA-MN. However, as you can see in Fig. 2(b), if the nAAAL reposes trust in the pAAAL, the MN just visited and then the authentication can be done without contacting the AAAH.

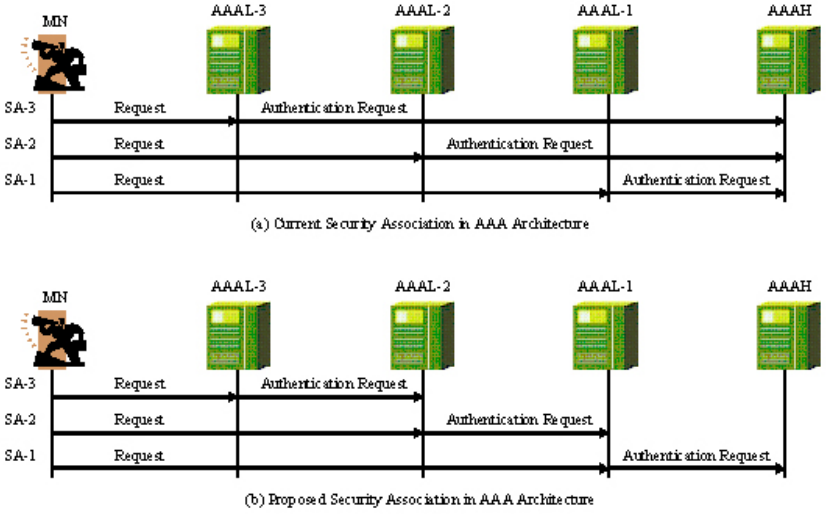


Fig. 2. Security Association in AAA Architecture

3 The Proposed Session Key Forwarding Scheme

In this section, we describe our session key forwarding scheme based on AAA architecture. The proposed scheme provides a possible solution for AAA based keying problems. As mentioned in [13], the inter-domain handoff scenario would require that session keys for FAs-MN would be different from the top level keys generated as parts of AAAs-FAs. The session keys calculation time and key delivery time are increased when the MN is far away from the MN's AAAH. In consequence, the authentication time is increased and then the handoff delay is increased.

In the proposed scheme, we need to make assumptions and describe the scenario. we assume as follows: (1) Secure channels are assumed to connect AAA servers (AAAH, AAALs) and clients (HA, FAs). (2) AAA servers and clients

have the function to generate and deliver the AAA messages. (3) AAA servers have a list of their neighbor to forward session keys with them. (4) AAALs keep the session key for its FA-MN. The session keys forwarding scenario is as follows: When a MN comes to a new foreign network from a previous foreign network, the MN needs session keys (HA-MN, nFA-MN) to make a new SA. The session key for HA-MN has been generated during the MN started up on a home network if the pAAAL provides the key, so it is no need to generate a new one each time the MN visits to the new foreign network. Therefore, the MN only needs the session key for nFA-MN. It minimizes an additional round trip through the internet when the MN moves into new foreign domains, and enables smooth handoff.

3.1 Messages Flow

In Fig. 3, we describe how to obtain the session key for nFA-MN. (1) The MN visits the new foreign network and sends the Registration Request to the nFA. The authentication data (HMAC-MD5 hash) is encrypted with the session key shared between the pFA and MN. Along the Registration Request, the previous foreign network address should be attached. The address should be kept by the MN while it is communicating. (2) The nFA sends the AAA-MN Request to the nAAAL. (3) The nAAAL receives the AAA-MN Request and obtains the pAAAL address and then forwards the message to the pAAAL. (4) The pAAAL authenticates the message by verifying the hash value, and generates the nonce for the MN and nFA. (5) The AAA-MN Answer is generated in the pAAAL. The message includes a plain nonce and an encrypted nonce using the key shard between the pFA and MN and then the message is encrypted and signed using the key shared between the pAAAL and nAAAL. (6) Upon receiving the AAA-MN Answer from the pAAAL, the nAAAL decrypts it and forwards it to the nFA. (7) The nFA receives the plain nonce and generates the key. it also brings together the encrypted nonce with Registration Reply and then sends it to the MN. (8) Upon receipt of the message the MN decrypts and stores the contained session key for nFA-MN. The MN can now start using the services in the new foreign network.

3.2 Security Considerations

AAA architecture must ensure a number of security threats. In this section, We have analyzed three cases, which are namely non-repudiation, impersonation attack, and replay attack, in the proposed scheme. (1) Non-repudiation: A repudiation attack can occur whereby the sender of a message denies having send the information. In this paper, when a MN wants to obtain the connecting request to a new foreign network, the MN must use the session key for HA-MN. An AAAH validates the signature signed with the session key. If the verification holds, the MN cannot deny when the dispute occurs. (2) Impersonation attack: If an attacker tries to impersonate a legal user in a new foreign network, the attacker will not be successful because the attacker does not possess the session key for

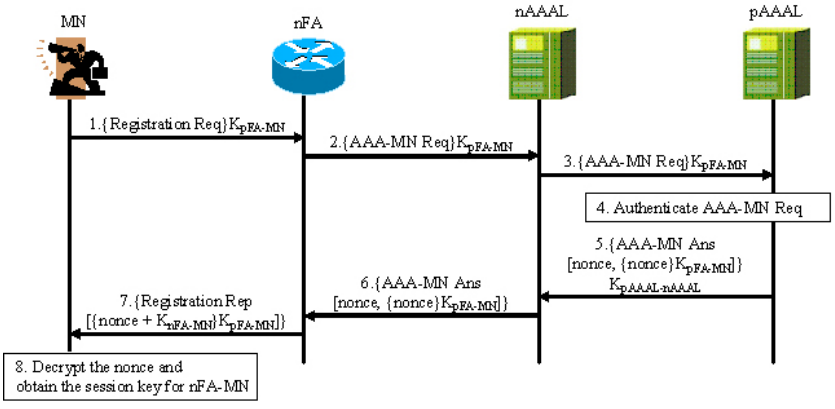


Fig. 3. Flow of total messages to obtain the session key for nFA-MN

pFA-MN. The effective key strength should be stated as a number of bits. If the key strength is N bits, the best currently known methods to recover the key require, on average, an effort comparable to 2^{N-1} operations of a typical block cipher. (3) Replay attack: If an attacker tries to replay the previous message sent by a legal MN, it will not be successful because of the invalid time-stamp. The expiration will be effectively verified in AAA servers.

4 Performance Evaluation

In this section, we do the performance evaluation to obtain the total authentication time according to the message flows (Figure 3) including the processing time needed in authentication operations, the transmission time on which the message is sent for wired and wireless links, and the processing time to authenticate a MN. The obtained total authentication time is used to calculate the MN's authentication failure rate.

4.1 Total Authentication Time

In order to evaluate the performance of our proposed scheme, we define the following notations:

- T_{MN-nFA} / $T_{nFA-nAAAL}$ / $T_{nAAAL-pAAAL}$ / $T_{pAAAL-nAAAL}$ / $T_{nAAAL-nFA}$ / T_{nFA-MN}: The transmission time between the MN and nFA / the nFA and nAAAL / the nAAAL and pAAAL / the pAAAL and nAAAL / the nAAAL and nFA / the nFA and MN, respectively.
- P_{MN} / P_{nFA} / P_{nAAAL} / P_{pAAAL}: The processing time at the MN / nFA / nAAAL / pAAAL, respectively.
- AT_{pAAAL}: The authentication time at the pAAAL.

Table 1. System parameters

Bit rate		**Processing time**	
Wire/Wireless link	100 Mbps / 2 Mbps	pAAAL/nAAAL/AAAH	0.5 msec
Propagation time		pFA/nFA/MN	0.5 msec
Wire/Wireless link	500 μsec / 2 msec	3DES	0.5 msec
Data size		MAC	0.5 msec
Message size	256 bytes	AT (Authentication Time)	6.0 msec

We calculate times required for the performance evaluation using the following equations as above notations and the performance parameters used to analyze the scheme are listed in Table 1. Each value is defined based on [7,8,9,10,11].

First of all, the message transmission on wired links is in steps 2, 3, 5, and 6. Thus, $T_{total}^{wired} = T_{nFA-nAAAL} + (T_{nAAAL-pAAAL}) + (T_{pAAAL-nAAAL}) + T_{nAAAL-nFA}$ and the message transmission on wireless links is in steps 1 and 7. Thus, $T_{total}^{wireless} = T_{MN-nFA} + T_{nFA-MN}$. The processing time is represented as $P_{total} = P_{MN} + P_{nAAAL} + P_{pAAAL} + P_{nFA} + P_{MN}$. The authentication time AT is required in step 4. Hence the total authentication time AT_{total} is AT_{pAAAL}. Therefore, we obtain the total required time for the authentication completion. The total authentication time (T_{req}) is $T_{total}^{wired} + T_{total}^{wireless} + P_{total} + AT_{total}$.

4.2 Authentication Failure Rate

The T is a random variable of the time for a MN staying in the overlapped area and the T_{req} is the time required for the authentication completion. Hence, the probability ($\tilde{P}$) which the MN leaves the overlapped area before the required time (T_{req}) is represented as $\tilde{P} = Prob(T < T_{req})$, where we assume that T is exponentially distributed and we restrict this probability to a certain threshold (P_f). Thus, the authentication failure rate $\tilde{P} = Prob(T < T_{req}) = 1 - exp(-\lambda T_{req}) < P_f$. Here is λ the arrival rate of MN into the boundary cell and its movement direction is uniformly distributed on the interval $[0, 2\pi)$. Thus λ is calculated by the equation $\lambda = \frac{VL}{\pi S}$ [12]. Here V is the expected velocity for MN that varies in the given environment and L is the length of the boundary at the overlapped area assuming a circle with radius l, i.e. $L = (\frac{1}{6} \times 2\pi l) \times 2 = \frac{2}{3}\pi l$. The area of the overlapped space S is $2 \times (\frac{1}{6}\pi l^2 - \frac{\sqrt{3}}{4} l^2)$. Hence we obtain the authentication failure rate by T_{req} and λ.

$$l > \frac{4VT_{req}}{(2\pi - 3\sqrt{3})log(1/(1 - P_f))} \tag{1}$$

4.3 Numerical Results

Using above equations and the system parameters in Table 1, we compute the cumulative authentication latency and the authentication failure rate by the

increment of V. As you can see, Fig. 4 shows the result of the cumulative authentication latency. Our proposed scheme always shows the better performance than the standard AAA scheme due to decrease of the authentication message forwarding distance to the AAAH. The authentication latency is improved up to about 6.16%. Fig. 5 compares results for the probability of authentication failure between the standard AAA scheme and the proposed one. The probability of authentication failure is influenced by few factors that are the velocity of MN and the radius of a cell(l). The increase of MN velocity (V) means the handoff should be completed within relatively short period of time. If the MN moves faster than the regular speed, the handoff time may not be sufficient and consequently the authentication is failed.

The graph in Fig. 5 shows the authentication failure rate as a function of the radius of the cell. As the radius of the cell increases, the overlapped area

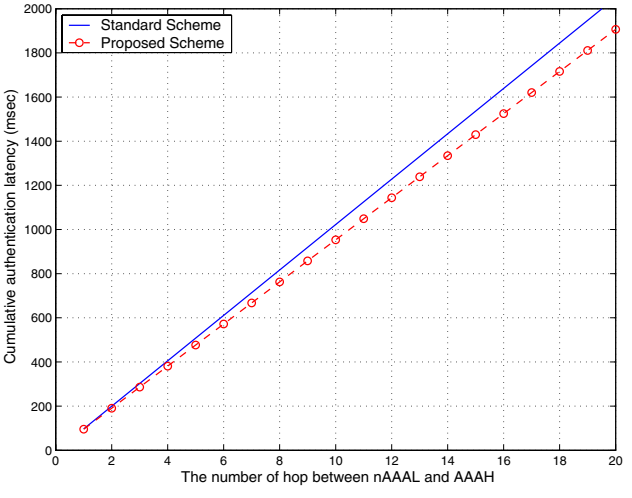


Fig. 4. The cumulative authentication latency

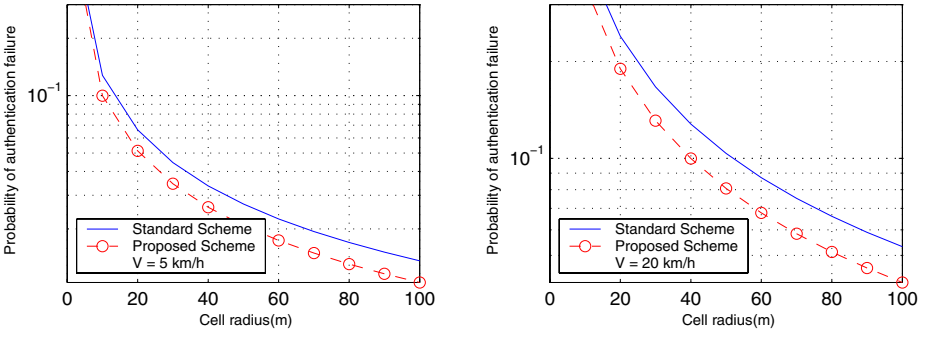


Fig. 5. The probability of authentication failure by the increment of V

becomes larger, so that it is easy to complete the handoff. As a result, the authentication failure rate decreases. When an authentication occurs with a MN which has a high velocity, it is difficult to complete the authentication, because the MN moves out of the overlapped area quickly. We calculate the cell size when the probability of authentication failure is 10% and the velocity of MN is 20km/h. By the equation (1), the minimum cell radius of the existing scheme is 52.7m, meanwhile the proposed one is 40.1m. Therefore, the proposed scheme is more efficient than the existing one in terms of cell radius and the efficiency is improved up to about 23.9%. As a result, the proposed scheme is overall superior to the existing one and even it supports more stable environments.

5 Conclusions

In this paper, we propose session key forwarding scheme based on AAA architecture. Currently, the standard AAA architecture causes long authentication delay because the AAAL must request the MN's authentication to its AAAH for verification. To solve this problem, the proposed scheme performs the session key forwarding to reduce the authentication delay with MN's AAAH. According to the analytical model and the comprehensive numerical results, our scheme shows the better performance in the cumulative authentication latency. Moreover, the proposed scheme is more efficient than the previous one in terms of cell radius up to about 23.9% with almost negligible signaling overhead. In the future, we will describe the requirements for the efficient session key management and the access control by policy based.

Acknowledgment

This study was supported by a grant of the Korea Health 21 R&D Project, Ministry of Health & Welfare, Republic of Korea. (02-PJ3-PG6-EV08-0001)

References

1. A.K. Arumugam, A. Doufexi, A.R. Nix, and P.N. Fletcher, "An Investigation of the Coexistence of 802.11g WLAN and High Data Rate Bluetooth Enabled Consumer Electronic Devices in Indoor Home and Office Environments", IEEE Transactions on Consumer Electronics, pp. 587-596, August 2003.
2. L. Salgarelli, M. Buddhikot, J. Garay, S. Patel, and S. Miller, "The Evolution of Wireless LANs and PANs - Efficient Authentication and Key Distribution in Wireless IP Networks", IEEE Personal Communications on Wireless Communications, pp. 52-61, December 2003.
3. S. Shieh, F. Ho, and Y. Huang, "An Efficient Authentication Protocol for Mobile Networks", Journal of Information Science and Engineering, pp. 505-520, January 1999.
4. V. Narayanan, N. Venkitaraman, H. Tschofenig, G. Giaretta, and J. Bournelle, "draft-vidya-mipshop-handover-keys-aaa-02.txt", Internet-Draft, April 2006.

5. S. Farrell, j. Vollbrecht, P. Calhoun, and L. Gommans, "AAA Authorization Requirements", RFC 2906, August 2000.
6. C. de Laat, G. Gross, L. Gommans, J. Vollbrecht, and D. Spence, "Generic AAA Architecture", RFC 2903, August 2000.
7. J. Xie and I.F. Akyildiz, "An optimal location management scheme for minimizing signaling cost in Mobile IP", ICC 2002, pp. 3313 - 3317, May 2002.
8. J. McNair, I.F. Akyildiz and D. Bender, "Handoffs for real-time traffic in Mobile IP version 6 networks", GLOBECOM 2001, pp. 3463-3467, November 2001.
9. A. Hess and G. Shafer, "Performance Evaluation of AAA/Mobile IP Authentication", Proceedings of 2nd Polish-German Teletraffic Symposium (PGTS02), September 2002.
10. J. McNair, I.F. Akyldiz, and M.D. Bender, "An inter-system handoff technique for the IMT-2000 system", INFOCOM 2000, pp. 203-216, March 2000.
11. Wei Dai, "The most commonly used cryptographic algorithms Benchmarks", http://www.eskimo.com/ weidai/benchmarks.html, Accessed on January 2006.
12. R. Thomas, H. Gilbert, and G. Mazziotto, "Influence of the moving of the mobile stations on the performance of a radio mobile cellular network", Proceedings of 3rd Nordic Seminar, pp. 1-9, September 1988.
13. M. Nakhjiri, M. Parthasarathy, J. Bournelle, H. Tschofenig, and R. Marin Lopez, "AAA based Keying for Wireless Handovers: Problem Statement", draft-nakhjiri-aaa-hokey-ps-02, Internet-Draft, May 2006.

Dynamic Anchor Based Mobility Management Scheme for Mobile IP Networks*

Jae-Pil Yoo and Kee-Cheon Kim**

School of Computer Science & Engineering. Konkuk University,
Seoul, Korea, +82-2-450-3518
{willow, kckim}@konkuk.ac.kr
http://mbc.konkuk.ac.kr

Abstract. In this paper, we introduce a dynamic anchor based mobility management scheme for mobile networks, in which different hierarchies are dynamically set up for different users and the multiple tunnels are converged into a single tunnel when lifetime refreshes without excessive packet loss. To justify the effectiveness of our proposed scheme, we made an analytic model to evaluate the signaling cost and handoff delay compared with legacy schemes. Our performance result shows that the proposed dynamic anchor based mobility management scheme can reduce the system signaling cost and has shorter handoff delay under various scenarios.

1 Introduction

The tremendous growth of wireless communications and the technological advance of mobile terminals, for example, laptop and PDAs, encourage a growing demand for mobile and nomadic computing. Mobile IP [1][2] is one of the dominating protocols that provide the mobility support in the Internet. Mobile IP uses two addresses and two support agents to support host mobility. One address and an agent are used for it identification, and the other address and the agent are used for its routing. We say such addresses and agents as home address, home agents, care-of address and foreign agent respectively. Even though mobile IP keeps its network and upper layer connection† when moves between networks, it has small period of time that cannot transfer any data during handover. Such duration of time delay, so called 'handover signal delay', is in proportion to the distance between mobile host and its home agent and the frequency of the movement of mobile host. Since we cannot shorten geographical distance between mobile nodes and home agents, many other approaches to overcome the handover signal delay were devised [3-19].

The local registration scheme using FA hierarchy is the typical approach to overcome the handover delay. In the local registration using FA hierarchy, mobile host selects the foreign agent for registration target closer than its home agent to reduce round trip time from mobile node to home agent. Regional Registration [20], HMIPv4 (HUT)[21], TeleMIP [22] schemes are based on local registration method.

* This research was supported by the Brain Korea 21 Project.
** Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 27 – 38, 2006.
© Springer-Verlag Berlin Heidelberg 2006

However, Local registration scheme using FA hierarchy is not always superior to basic mobile IP. The GFA (Gateway Foreign Agent) that is a domain gateway in FA hierarchy, acts as a crossover node to all the downward links. Within a domain, the GFA must handle all the signals and traffics for mobile nodes. If packets and signals are concentrated on the GFA, overall system performance could be decreased or failed by the GFA crash. Also, we cannot overemphasize the multiple tunnel problem of FA hierarchy. In most cases, local registration schemes have two or more tunnels for packet delivery. IP-in-IP tunneling requires a series of process such as encapsulation and de-capsulation. Especially, FAs in a FA hierarchy must process encapsulation and de-capsulation steps all the time for multiple tunnels. So, the depth of multiple tunnels is important factor for system performance.

In this paper, we introduce a dynamic anchor based mobility management scheme. Although the anchor could be a local HA like GFA in regional registration, the anchor is set dynamically based on system threshold value using the distance and timing value. Therefore, our system has no fixed or pre-defined hierarchy information or any restriction of geographical location for network configuration. These dynamic anchor allocation characteristics enable network to balance the load and to make a robust system in comparing with geographically fixed FA hierarchy system.

The proposed scheme uses lifetime reflection scheme to synchronize the lifetime values among mobility agents. In many cases, local registration scheme needs additional registration messages to synchronize lifetime value in all the mobility agents from the MN to HA. In our system, we minimized the number of signal messages for synchronization among mobility agents using lifetime reflection scheme.

2 Dynamic Anchor Based Mobility Management Scheme

In this section, we describe an architecture and process of the proposed system. The proposed system updates location information locally based on the anchor. The anchor, a kind of mobility agent has function of HA and FA in MIP, acts as local HA for mobile node. The anchor in the proposed scheme is selected among FAs that MN has traveled. Therefore, the anchor is not fixed geographically and created or disappeared by time and location. MN can decide whether to do a local registration or home registration based on the 'threshold' value, where the threshold is a critical value used for re-creating an anchor. In our evaluation model, we set the threshold value based on the distance between the anchor and the FA. Network administrator can set the suitable threshold value to MN for better performance

Dynamic anchor based mobility management is shown in Fig. 1. In order to understand the mobility management scheme, we can assume that MN moves from FA1 to FA7 with threshold value of 3, where the threshold values means the distance from the anchor to FA to which MN is attached. When the MN attaches to the network for the first time, it performs the home registration through FA1 to HA. MN sets the current FA (FA1) as its anchor node for a future location update. After this, MN can do local registration when it enters a new network until the distance between the anchor and new network is equal to the threshold value. As shown in Fig. 1., MN performs a local registration with the anchor(FA1) when moves from FA1 to FA2 and FA3, where the FA1 becomes the local HA reference point to other FAs. When MN moves into FA4 coverage area, it sets current FA4 as its anchor by threshold value.

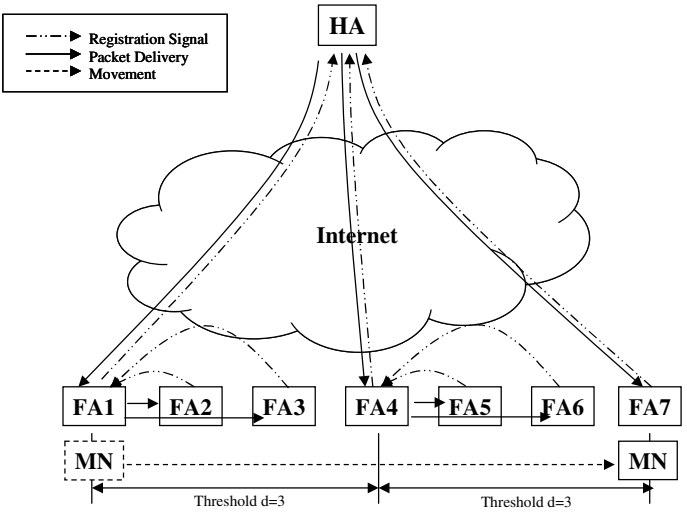


Fig. 1. Dynamic Anchor based mobility management operations

Movement into FA4 of the MN makes threshold value as '0'. So, the MN can perform a local registration until it reaches FA6 because the distance between the anchor and the FA does not exceed the threshold value. We can guess easily that the cost of registration would be bigger if the threshold is very small by frequent signals. On the other hand, Big threshold value would create a longer location update delay because of the long distance between the anchor and the FA. Therefore, we need a proper threshold value for optimal performance for mobile network.

Packet transmission after registration is as following. HA always knows MN's CoA as an Anchor's CoA. Therefore, all the packets destined to MN from CN are arrived at the anchor via the HA. The anchor that receives tunneled data from CN prepares another tunnel from the address of the anchor to CoA of FA in its binding list entry for MN after performing decapsulation. Therefore, there is one tunnel between HA-MN, when MN co-exists within a same area with the Anchor and there are two tunnels between HA-MN, when MN is out of the Anchor area.

MIP's location information is kept by a soft-state based timer. The proposed scheme use this timer except lifetime synchronization method among HA, FA and MN. When MN moves into FA2, it performs a local registration to FA1 as its anchor node. After that, FA1 handles a local registration signal from MN via FA2. Every local registration performed without HA produces lifetime inconsistency problem between HA and other nodes. We can assume that MN plies FA2 and FA3 when threshold value is 3. This time, because local registration is processed by FA1 that is an Anchor of MN, The registration message does not reach to HA. Therefore, lifetime has been synchronized among the anchor, FA, and MN, but the lifetime value in HA does not consistent with the one in anchor or FA. Lifetime expiration means the deletion of the relevant MN's entry in HA's binding list and the data that CN has sent to MN are discarded at the home network of MN.

In order to prevent such data loss, we can consider HA-Anchor-nFA-MN registration per network movement. In this case, longer processing time in many mobility agents and long round trip distance from MN to HA makes always worse result than MIP's registration. Therefore, we need to consider a suitable scheme satisfying the lifetime synchronization from HA to MN and minimal registration delay when MN performs local registration.

Our scheme uses lifetime reflection scheme to solve the above problems. The 'reflection' in lifetime reflection scheme means that the anchor node reflects the remained lifetime value to the source node that invoked local registration to synchronize the lifetime with HA. For example, we can assume network environment with 300 seconds of lifetime value in HA, Anchor, nFA, and MN, where MN moves into another network per 120 seconds. When MN without lifetime reflection scheme performs local registration, the lifetime value of the anchor, FA, and MN would be reset by 300 seconds. But in case of lifetime reflection scheme, the Anchor sends registration reply using the current MN's remained lifetime in its binding list. Therefore, after all the lifetime values in FA, MN, and the anchor would be synchronized to 180 seconds. If MN moves once more, all the lifetime value in the anchor, nFA, and MN are set to 60 seconds by lifetime reflection scheme mentioned above. At that time, MN must perform a home registration from MN to HA via FA and the anchor to prevent lifetime expiration.

3 Performance Modeling

In this section, we developed a performance model to evaluate the proposed scheme. On our performance model, We made average registration cost function per movement and average handoff delay function per movement of basic MIP and the proposed system respectively using 'Random Walk Model' as a mobility simulation model. And we derived various results from those function using different parameters that can influence in the performance.

We use the 2-dimensional grid area made of cells for mobility area. HA, FA, MN can move freely and send/receive packets to and from in the basic unit cell shown as Fig. 2. Basis unit area on the grid means a cell and it means a sub-network of a real world. Cells do not overlap and just one mobile agent is existed on each cell. In this cell based grid area, the distance between cells means the distance between mobility nodes, for example, MN, FA, HA…etc. In case of FA and MN, we assume that they always exist in the same cell because they share direct link in our model. We use a cell movement count to present the distance between cells. For example, the distance between two cells with coordinates (x1, y1) and (x2, y2) is $| x2 - x1 | + | y2 - y1 |$. We apply Random Walk Model to calculate the mobility pattern. Random walk model has an equal probability of movement for all 4 sides of the cell in 2 dimensional plane, where the walk model is widely used for modeling in many other papers.[23][24][25]

3.1 System Modeling

Circled 'A' in Fig. 2. Means Mobility Nodes. We set the anchor node in the proposed system on the center of the grid area, so it has distance of '0'. The MN moves from

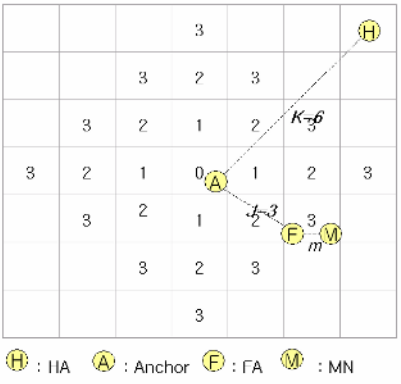


Fig. 2. Grid area for system modeling with Random Walk Model

the center to anywhere in the grid area freely but cannot move out of the grid area. The grid area size is restricted to a limited value. It is the value of the threshold number. So the grid area size is decided by the distance between FA and anchor, where the distance means the threshold value. It is reasonable because MN and FA always co-exist within a same cell.

Table 1. Shows parameters used in our modeling. In order to derive a reasonable performance, we made link property; $\delta\Gamma, m, C_A$, and K as input parameters. And the other parameters are computed by probability in Random Walk Model using random variables.

Table 1. Parameters used on modeling

Parameter	Explanation
C_A	Signal Processing Cost of Mobility Agent
$\delta\Gamma$	Proportionality constant between the transmission cost of the wired link
C_{ha}	Transmission cost between HA-Anchor
C_{af}	Transmission cost between Anchor-FA
C_{fm}	Transmission cost between FA-MN
C_{hf}	Transmission cost between HA-FA
k	Distance from HA to Anchor
j	Distance from Anchor to FA
m	Proportionality constant between the transmission cost of the wireless link
d	Threshold value for dynamic anchor

3.2 Cost Function for Our Scheme Under Random Walk Model

First, we find out two cases of local registration of the proposed system to get registration cost. Local registration is classed into two cases by registration cost as shown in Fig. 3.by registration cost. Fig. 3. depicts the case of local registration (Handoff) with an anchor and case of returning to the anchor

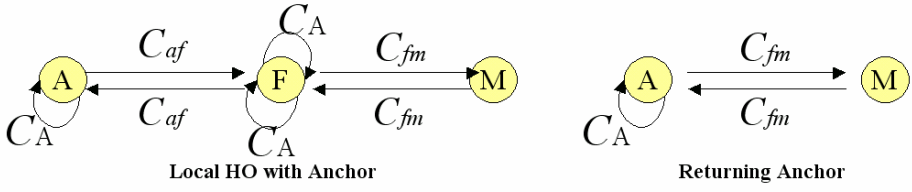


Fig. 3. Local handoff case for Dynamic Anchor scheme

So, we can express local registration cost as follows using parameters in Table 2.

$$\begin{cases} 2C_{af} + 3C_{\mathrm{A}} + 2C_{fm} : \text{Local Regisration using Anchor} \\ C_{\mathrm{A}} + 2C_{fm} \quad\quad\quad : \text{When returns to the anchor} \end{cases} \tag{1}$$

We can rewrite (1) as

$$\begin{cases} 2(j+m)\delta\mathrm{T} + 3C_{\mathrm{A}} \\ 2m\delta\mathrm{T} + C_{\mathrm{A}} \end{cases} \tag{2}$$

We normalize the above (2) using random variables. $Cost^{n}_{LHO}$ expresses average cost of local HO after n-th movement, where P_{RA} is a probability that MN returns to the anchor.

$$Cost^{n}_{LHO} = P_{RA}(C_{\mathrm{A}} + 2m\delta\mathrm{T}) + (1 - P_{RA})[2(j+m)\delta\mathrm{T} + 3C_{\mathrm{A}} \tag{3}$$

In order to formulate P_{RA}, we transform the movement probability pattern of Random Walk Model into transition matrix. Each element $p^{n}_{i,j}(0 \le i, j \le d)$ in the matrix P represents the probability of a MN moving from a distance-i cell to a distance-j cell in exactly n movements. In case of $P^{3}_{0,2}$, It represents the probability of a MN moving from a center to distant-2 cell in 2 movements. The matrix has $(d+1)\times(d+1)$ dimension by same reason that the grid area size is decide by the distance between FA and anchor, where d represents the threshold value. Fig. 4. depicts the matrix elements of matrix P^{1}_{d}.

$$P^{1}_{d} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & \cdots & a_{0,d} \\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & a_{0,d} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{d-1,0} & a_{d-1,1} & a_{d-2,1} & \cdots & a_{d-1,d} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

We can obtain rule (5) by expanding Random Walk Model movement pattern. (5) shows the element value given *i, j* conditions, where *'i'* is the starting cell and *'j'* is the destination.

$$a_{i,j} = \begin{cases} 1 & : i = 0, j = 1 \\ 2i + 1/4i & : i > 0, j = i + 1 \\ 2i - 1/4i & : i > 0, j = i - 1 \\ 0 & : else \end{cases} \tag{5}$$

This transition matrix base on Random Walk Model has rules of Markov Chain.[26] So, we can get *n-th* result by (6)

$$P_d^n = P_d^1 \times P_d^{n-1} \tag{6}$$

So, we can rewite local registration using dynamic anchor as follows

$$Cost_{LHO}^n = p_{0,0}^n (C_{\mathrm{A}} + 2m\delta\Gamma) + \sum_{j=1}^{n} p_{0,j}^n [2(j+m)\delta\Gamma + 3C_{\mathrm{A}}] \tag{7}$$

Besides, MN performs home registration when the distance from the anchor to FA reaches the threshold value. Home registration cost wound be (8)

$$Cost_{\mathrm{home}} = 2C_{hf} + 3C_{\mathrm{A}} + 2C_{fm} \tag{8}$$

We can rewrite (8) as follows with the same method from (2) to (3)

$$Cost_{\mathrm{home}} = 2(k+m)\delta\Gamma + 3C_{\mathrm{A}} \tag{9}$$

Because there is exacly one local registration per movement of the MN and one home registration every *d* movement, so the average registration cost per movement after *d* movement is expressed as

$$\frac{Cost_{\mathrm{home}} + \sum_{n=1}^{d} Cost_{LHO}^n}{d} \tag{10}$$

Finally, derivation of (10) can be

$$\frac{2(k+m)\delta\Gamma + 3C_{\mathrm{A}} + \sum_{n=1}^{d} [p_{0,0}^n (C_{\mathrm{A}} + 2m\delta\Gamma) + \sum_{j=1}^{n} p_{0,j}^n [2(j+m)\delta\Gamma + 3C_{\mathrm{A}}]]}{d} \tag{11}$$

3.3 Cost Function for Mobile IP Under Random Walk Model

Similarly with 3.2, MIP registration is classfied into two cases.

$$\begin{cases} 2C_{hf} + 3C_{\mathrm{A}} + 2C_{fm} : MIP\ registration \\ C_{\mathrm{A}} + 2C_{fm} \qquad\quad : When\ returns\ to\ the\ HA \end{cases} \tag{12}$$

We can define formula (12) as follows with probability case. $Cost_{n-\mathrm{home}}$ denotes average registration cost after n-th movement. In (13), P^n_{home} denotes the probability of MN moving to home network.

$$Cost_{n-\mathrm{home}} = p^n_{\mathrm{home}}(C_{\mathrm{A}} + 2m\delta\Gamma) + (1 - p^n_{\mathrm{home}})[2(k+m)\delta\Gamma + 3C_{\mathrm{A}}] \tag{13}$$

In order to present P^n_{home} as a numerical fomula, we reuse transition matrix P in 3.2. $P^n_{0,3}$ in matrix $P^n_{i,j}$ means that MN is far from center by distance of '3'. When k=3 and $P^n_{0,3} \neq 0$, MN has the probability of returing home. To calculate the average probability of returning home, we used grid calculation on the grid area in Fig. 2. We can see the counts of cell that has distance of *'d'* from center are $4 \times d$. HA could be exactly in one of these cells, so the average HA existence probability is $\frac{1}{4d}$. By these rules, (13) could be rewrited as follows.

$$P^n_{\mathrm{home}} = \frac{p^n_{0,k}}{4k}(k \leq n) \tag{14}$$

So, the $Cost_{n-\mathrm{home}}$ is

$$Cost_{n-\mathrm{home}} = \frac{p^n_{0,k}}{4k}(C_{\mathrm{A}} + 2m\delta\Gamma) + (1 - \frac{p^n_{0,k}}{4k})[2(k+m)\delta\Gamma + 3C_{\mathrm{A}}](k \leq n) \tag{15}$$

Finally, we can derive per movement average MIP location update cost using the following function.

$$Cost_{MIP} = \frac{\sum_{n=1}^{d}[\frac{p^n_{0,k}}{4k}(2m\delta\Gamma + C_{\mathrm{A}}) + (1 - \frac{p^n_{0,k}}{4k})\{2(k+m)\delta + 3C_{\mathrm{A}}\}]}{d} \tag{16}$$

4 Performance Evaluation

In this section, we evaluate the model in different environments using the modelling functions we have derived in section 3. We set the following network parameters to apply different kind of network environment on the functions.

Table 2. Parameters used in modeling

Parameter	value
C_{A}	0.05, 0.5(ms)
$\delta\Gamma$	0.1, 1(ms)
k	1,4,7,10,13,17,20(hops)
m	$\delta\Gamma$ *5(ms)

Table 2. shows parameters used for our evaluation, where parameter with $C_A = 0.05$, $\delta\Gamma = 0.1$ represents fast network environment and parameter with $C_A = 0.5, \delta\Gamma = 1$ represents for relatively slow network environment.

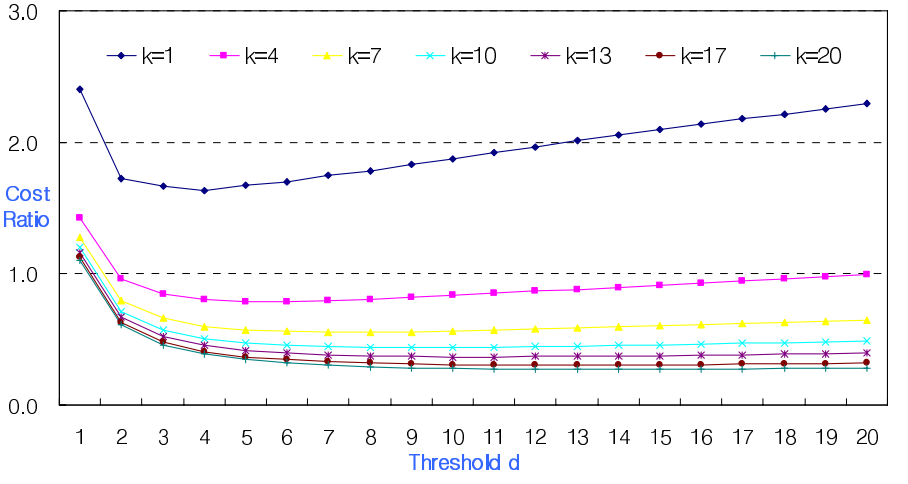


Fig. 4. Ratio of registration cost in random walk model ($\delta\Gamma = 0.1, C_A = 0.05, m = 0.5$)

Fig. 4. depicts average signal cost ratio per movement of the proposed scheme to legacy mobile IP under condition of relatively fast network environment $(C_A = 0.05, \delta\Gamma = 0.1, m = \delta\Gamma \times 5)$. Since the values of Y axis mean the ratio of our signal cost to legacy mobile IP, We can evaluate relative performance on the basis of '1'. As shown in Fig. 4. , The signal cost of the proposed scheme is always superior to legacy mobile IP when *k>=4* and *d>1* . Our scheme is inferior to legacy scheme only when the MN is near HA due to additional signals such as local registration per movement. But, When MN moves far from HA with *threshold=d>2*, registrations to far HA with high cost are substitute as local registration with low cost. Therefore, When MN is far from HA in some degree(*k>=4*) and threshold value is suitable(*d>1*), we can expect signal cost reduction effect using the proposed scheme.

Fig. 5. depicts handoff delay ratio of the proposed scheme to legacy scheme. The handoff delay functions can be derived from registration cost functions. We can substitute transmisstion delay for transmission cost and processing delay for processing cost to derive delay measure. Home registration in our scheme does not invoke packet loss when its handoff because MN can always receive buffered data from its anchor. Therefore, the handoff delay of the proposed schme is as follows.

$$\frac{\sum_{n=1}^{d}[p_{0,0}^{n}(C_A + 2m\delta\Gamma) + \sum_{j=1}^{n} p_{0,j}^{n}[2(j+m)\delta\Gamma + 3C_A]]}{d} \tag{17}$$

In case of mobile IP, all the registratins invoke packet loss. For that reason, handoff delay of the mobile IP is the same with (16). Similarly with cost results as shown

in Fig. 4., Our scheme shows dorminat result over legacy mobile IP scheme except $k \neq 1$. Even though the ratio value would excess '1' when k goes infinite value by gradient of curves in Fig. 5, k values would not excess '20' when we assume that hop counts of common network session between two nodes in real world is less than '20'.

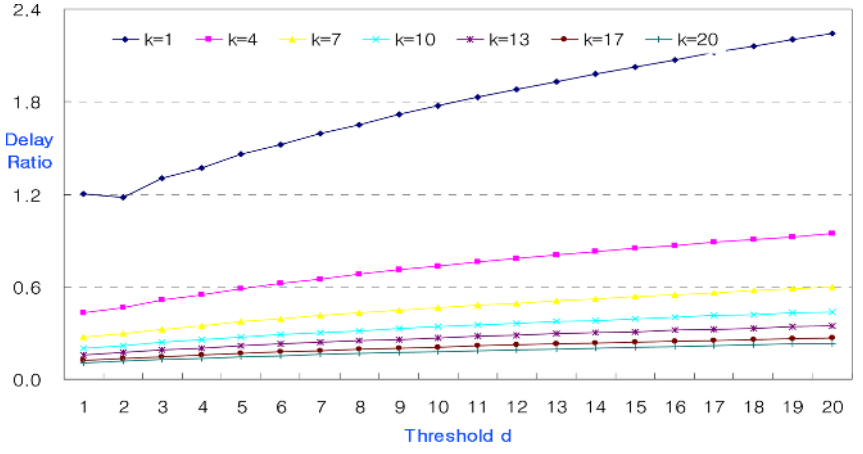


Fig. 5. Ratio of HO delay in random walk model ($\delta\Gamma = 0.1, C_A = 0.05, m = 0.5$)

Fig. 6. and Fig. 7. shows performance result with relatively slow network environment. When we increase input parameter considerably such as tranmission cost constant or processing cost constant, the signal cost shows that our scheme does not have overwhelming result over legacy mobile IP scheme at a glance. But, Fig. 7. shows

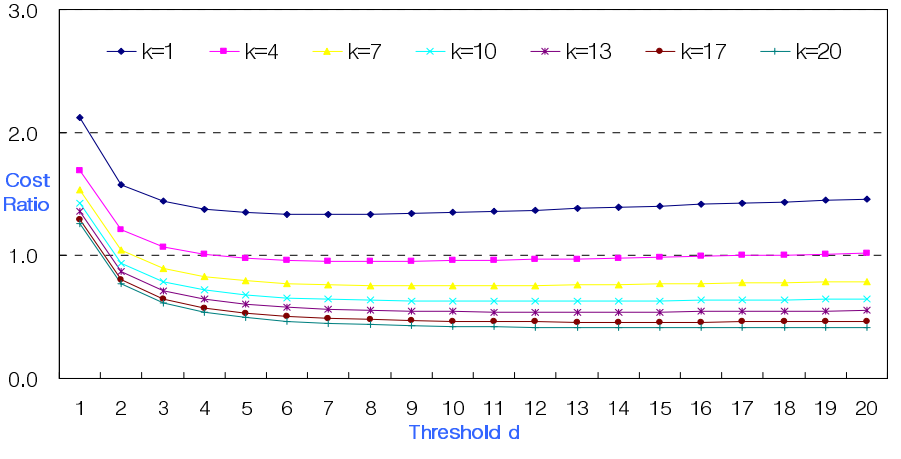


Fig. 6. Ratio of registraton cost in random walk model ($\delta\Gamma = 1, C_A = 0.5, m = 5$)

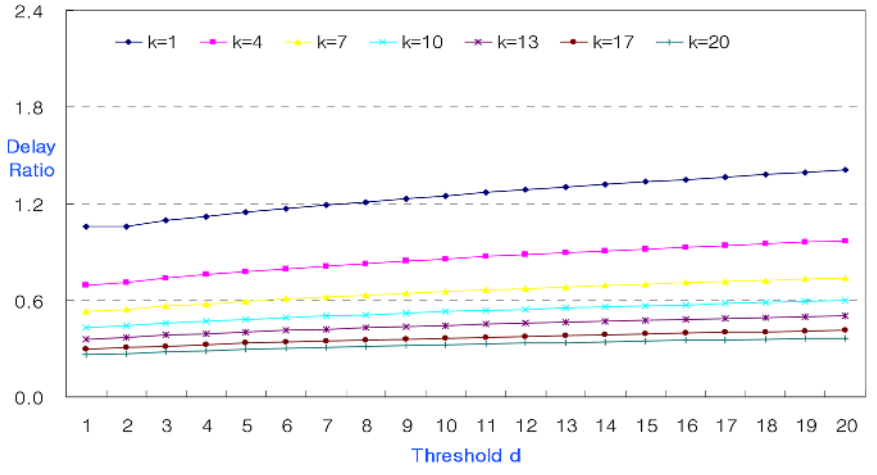


Fig. 7. Ratio of HO dealy in random walk model ($\delta\Gamma = 1, C_A = 0.5, m = 5$)

that our scheme has still very sound handoff delay result over legacy mobile IP scheme. It is reasnoable when we consider that local registration in our scheme is performed per movement. MN can utilize the various threshold values depend on the system performance aims. If MN wants low delay, small value of 'd' would satisfy the system aims. If MN wants low signal loads, more than '3' values would satisfy the system aims. In our performance evaluation, Threshold value of $3 \le d \le 7$ produces sound results when we consider average signal loads and delay synthetically.

5 Conclusions

In this paper, we proposed dynamic anchor based mobility management for improving Mobile IP performance. The scheme has following characteristics: reduced signal cost, reduced handoff delay, balanced-load on anchor nodes, compared with basic MIP. Our modeling results show that the proposed scheme has much better performance over basic MIP when the distance between the MN and the HA is far away.

References

1. J. Solomon, "Applicability Statement for IP Mobility Support", IEFT RFC, October 1996
2. C.E. Perkins, "IP Mobility Support", RFC 3344, August 2002
3. H. Chaskar, Editor, "Requirements of a QoS Solution for Mobile IP", IETF Internet-Draft, draft-ietf- mobileip-qos- requirements- 03.txt, July 2002.
4. C.E. Perkins and K.Y. Wang, "Optimized smooth handoffs in Mobile IP," IEEE International Symposium on Computers and Communications, pp. 340 -346, July 1999.
5. A. Campbell, J. Gomez, C-Y. Wan, S. Kim, "Cellular IP", draft-ietf-mobileip-cellularip-00.txt, January 2000
6. Edwards G, Suryakumar N (2003), Cellular IP Performance , IEEE Wireless Communication and Networking, 3:2081-2085

7. Campbell A T, Gomez J, Kim S, Valko A G, Wan C Y, Turanyi Z R (2000), Design, Implementation and Evaluation of Cellular IP, EEE Personal Communications, 7: 42-49
8. Campbell A T, Gomez J, Kim S, Wan C Y, Turanyi Z R, Valko G (2002), Comparison of IP Micromobility Protocols, IEEE Wireless Communications, 2 - 12
9. R. Ramjee, T. La Porta,S. Thuel, K. Varadhan, L. Salgarelli, "IP micro-mobility support using HAWAII", Internet Draft, Jun 1999
10. R. Whsieh, Z.G. Zhou, and A. Seneviratne, "S-MIP: a seamless handoff architecture for mobile IP," Proc. IEEE INFOCOM, vol. 3, pp. 1774 - 1784, 2003.
11. IETF Mobile IP Working Group, http://www.ietf.org/ html.charters/ mobileip-charter.html
12. K. El Malki et al., "Low Latency Handoffs in Mobile IPv4," Internet draft, draft-ietf-mobileip-lowlatency-handoffs-v4-11, Oct. 2005.
13. C. Blondia, O. Casals, Ll. Cerda, N. Van den Wijngaert, GWillems, P. De Cleyn, "Performance Comparison of Low Latency Mobile IP schemes", Proceedings of WiOpt'03 INRIA Sophia Antipolis, France 2003, pp. 115-124.
14. C. Blondia,O.Casals, Ll.Cerda, N. Van den Wijngaert, G Willems, P. De Cleyn, "Low Latency Handoff Mechanisms and their implementation in an IEEE 802.11 Network". ITC 2003.
15. A. Flademuller and R. De Silva. "The effect of Mobile IP handoffs on the performance of TCP", Mobile Networks and Applications, Vol. 4, No. 2, pp 131-135, May 1999.
16. J. McNair et al., A survey of cross-layer performance enhancements for Mobile IP networks, Computer Networks 49 (2005) 119. 146
17. Eva Gustafsson, Annika Jonsson, Charles E. Perkins, "Mobile IP Regional Registration", Internet Draft draft-ietf-mobileip-reg- tunnel-01.txt , November 2005
18. "Hut-dynamics." http://www.cs.hut.fi/Research/
19. S. Das. A. Misra, P. Agrawal. And S. K. Das, "TeleMIP:telecommunications-enhanced mobile IP architecture for fast intradomain mobility", IEEE Pers. Commun, vol. 7 , pp, 50-58, Aug.2000
20. J.J. Garcia-Luna-Aceves, E. Madruga, A multicast routing protocol for ad-hoc networks, in: IEEE INFOCOM, 1999, pp. 784.792.
21. I. Rubin, C. Choi, Impact of the location area structure on the performance of signaling channels in wireless cellular networks, IEEE Communications Magazine (1997) 108.115
22. M. Zonoozi, P. Dassanayake, User mobility modeling and characterization of mobility patterns, IEEE Journal on Selected Areas in Communications 15 (7) (1997) 1239. 1252.

Forest Search: A Paradigm for Faster Exploration of Scale-Free Networks*

Yuichi Kurumida, Hirotaka Ono, Kunihiko Sadakane, and Masafumi Yamashita

Department of Computer Science and Communication Engineering,
Kyushu University, Japan
kurumida@tcslab.csce.kyushu-u.ac.jp,
{ono, sada, mak}@csce.kyushu-u.ac.jp

Abstract. Scale-free networks naturally model wide area networks like the WWW. We consider the problem of fast exploration of huge scale-free networks using small memory space. Although there are many search algorithms for exploring an unknown graph, they require much space or time. For example, the depth first search requires some memory for all the nodes in the worst case, and the average number of steps in the random walk is $O(n^3)$, where n is the size of the graph. Under assumptions reflecting WWW applications, we propose a new exploration paradigm called *forest search* particularly designed for scale-free networks, and theoretically evaluate its space complexity. We also demonstrate its superiority over random walk based search algorithms by conducting simulations.

1 Introduction

Given an unknown graph, we in this paper consider the problems of (1) finding a path connecting two given nodes, and (2) traversing all the nodes. There are of course many search paradigms such as the depth first search (DFS) and the random walk, which are designed for general and abstract graphs, defined independently of concrete applications. Although these paradigms are widely applicable because of their abstract nature, they may not behave very well in real applications. For example, DFS requires some space for all the nodes in the worst case and the random walk requires $O(n^3)$-time on average, where these space and time are no longer moderate. We however may be able to come up with a better search paradigm by shifting and restricting our attention to a more concrete model.

The aim of this paper is to propose a new algorithm paradigm called the *forest search* for constructing search algorithms that guarantee good behaviors particularly on scale-free networks such as the WWW. The model we have in mind is the following:

1. A graph models the communication network of a distributed system, where nodes and edges represent sites and communication links, respectively. A crucial assumption is that the communication network is a logical one.

* This research partly received financial support from Scientific research fund of ministry of Education, Culture, Sports, Science and Technology.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 39–50, 2006.
© Springer-Verlag Berlin Heidelberg 2006

2. Each node has a unique identifier (ID), but the ID is unknown until the node is visited.
3. Each node identifies the communication links incident to it, but does not know who is adjacent to at the other end of a link.
4. An agent can memorize some node ID's, and jump directly to the node with an ID stored in the memory (imagine bookmarks of web browsers).
5. Our criterion is the number of steps of the agent to achieve the goal, and we do not consider the complexity to examine the neighbor nodes.

The agent is initially placed at an initial node. In the search algorithm, the agent moves on the graph nodes according to the link information. By Assumption (4) above, the agent may jump to a non-adjacent node, as there may be a virtual link. Note that this assumption can be removed at the cost of increasing the number of steps.

As touched in above, a paradigm for solving the problems is DFS on the graph. To keep the search path away from loops, we however need to memorize some information such as the visited node ID's or spanning trees [15,11,12]. Another paradigm is a random walk on the graph. Although a random motion of the agent requires less space, the worst case time complexity, i.e., the maximum number of steps necessary to achieve the goal, cannot be bounded from above. This fact and that the average time complexity is $O(n^3)$ are big disadvantages of the random walk paradigm.

This paper proposes the forest search paradigm, which requires less space than DFS and less time than the random walk. It is a deterministic paradigm, and the termination of search is explicit. The forest search is applicable to any graph, but it effectively works especially on scale-free networks [4]. It is known that many real wide area networks such as the WWW are scale-free.

The forest search defines a spanning forest of the graph by using local topological information and traverses each of the trees in the forest using a constant memory. Since the forest is implicitly maintained, we do not need to memorize it. To traverse all the trees in the forest, we make use of DFS paradigm. As the space complexity is in proportion to the number of the trees, we prove that it is small for scale-free graphs. We also show by simulation that all the nodes of a scale-free graph with one million nodes can be partitioned into less than four trees on average. We also conduct simulations to demonstrate its superiority over random walk based search algorithms. The results show that the forest search is better than the random walk with respect to the average time complexity to visit all the nodes.

2 Preliminaries

2.1 Scale-Free, Small-World and Random Graphs

The degree distribution follows the power-law in many real-world networks [1,2,8]; i.e., the probability $P(d)$ that a node has degree d satisfies

$$P(d) \propto d^{-\gamma}.$$

A network whose degree distribution follows the power-law is sometimes called a *scale-free network*. In many real-world scale-free networks, $2 < \gamma < 3$.

Barabási and Albert proposed a model, which we name BA model, to generate scale-free networks [4]. In BA model, a scale-free network is generated by the following two rules:

(Growth) We first prepare the complete graph with m_0 nodes. At every time step, we add a new node v and connect it to $m(\leq m_0)$ distinct nodes u randomly chosen from the existing nodes with the probability defined below. (That is, m edges are added.)

(Preferential Attachment) The probability that a node u is chosen is $\deg(u)/\sum_{v\in V'}\deg(v)$, where V' is the set of the current nodes. After t steps, the numbers of the nodes and the edges are thus m_0+t and $m_0(m_0-1)/2+mt$, respectively.

BA model generates scale-free networks with degree distribution $P(d) \propto d^{-3}$ [5]. We denote by G_m^n a graph randomly generated by BA model.

WS model is proposed by Watts and Strogatz to generate small-world networks [16]. To generate a small-world network, we prepare a regular lattice graph which is constructed from a ring by adding edges to connect each node to its k_w neighbors. We then re-wire each edge with a given probability p_w, where "re-wiring an edge" means disconnecting one end of the edge and connecting it to a node uniformly chosen at random so as not to create self-loops nor multiple edges. The number of the edges is $nk_w/2$ and for $0.01 < p_w < 1$ the generated graphs show the small-worldness [4].

ER model is a model to generate random graphs proposed by Erdős and Rényi [9]. In this model, random graphs are generated as follows: We first prepare n isolated nodes, and then, for each pair of nodes, independently, we connect them with a given probability p_e. The expected number of the edges is thus $p_e n(n-1)/2$ and the distribution of degree d is

$$P(d) = \binom{n-1}{d} p_e^d (1-p_e)^{n-1-d},$$

which is binomial with the mean degree $\bar{d} \stackrel{\text{def}}{=} p_e(n-1)$. If $n \to +\infty$, this converges to the Poisson distribution

$$P(d) \to \frac{e^{-\bar{d}}\bar{d}^d}{d!}.$$

2.2 Random Walks on Graphs

A random walk on a finite graph is the process that repeats moving an agent on a certain node to one of its neighbor nodes with some probability. The random walk often means the standard random walk in which the agent moves to one of the neighbor nodes with the same probability [7]. That is, the transition matrix $P = (p_{u,v})$ is given by

$$p_{u,v} = \begin{cases} 1/\deg(v) & (v \in N(u)) \\ 0 & (\text{otherwise}). \end{cases}$$

The standard random walk hence uses only the degree information of the node on which the agent exists. The *hitting time* $H_G(u, v)$ is the expected number of steps necessary for the agent in u to reach v. The *hitting time* of graph G is defined by $H_G = \max_{u,v \in V} H_G(u, v)$. For the standard random walk, the hitting time (indeed the cover time) of any graph G is bounded by $(1 + o(1))\frac{4}{27}n^3$ [10], and the hitting time of a lollipop (a complete graph of $\lceil \frac{n}{2} \rceil$ nodes with a path graph of $\lfloor \frac{n}{2} \rfloor$ nodes) is $(1 - o(1))\frac{4}{27}n^3$ [7].

Ikeda et al. recently generalized the standard random walk to propose the *β-simple random walk* [13]. The transition matrix of the β-simple random walk is given as follows:

$$p_{u,v}^{(\beta)} = \begin{cases} \frac{\deg^{-\beta}(v)}{\sum_{w \in N(u)} \deg^{-\beta}(w)} & (v \in N(u)) \\ 0 & (\text{otherwise}) \end{cases}$$

where $\beta \in \mathbf{R}$. Note first that it uses not only $\deg(u)$ but also $\deg(v)$ of $v \in N(u)$. Note next that the β-simple random walk is the standard random walk when $\beta = 0$. When $\beta < 0$ (resp. $\beta > 0$), the agent tends to move to a large (resp. small) degree node. For $\beta = 0.5$, the hitting time of any graph is bounded by $O(n^2)$, which is optimal, since there exists a graph whose hitting time is $\Omega(n^2)$ for any transition matrix [13].

The β-simple random walk has further been extended to (k, l, β)-random walk, where k is the size of *tabu* list, by which a move to k previously-visited nodes is avoided, and l is the look ahead distance, by which the agent can view the nodes within distance l from the current node and can jump to the target node (if there is) [14].

3 Forest Search

We now present the forest search paradigm. The basic idea is as follows. Scale-free networks G have the following properties: there are many nodes with small degrees, while there are a quite small number of nodes (*hubs*) having large degrees compared with the number of nodes with small degrees, and the average distance L_G ($L_G = \sum_{u \in V} \sum_{v \neq u} \mathrm{dist}(u, v)/n(n-1)$, where $\mathrm{dist}(u, v)$ is the length of the shortest path from u to v) is also small. Because of the preferential attachment, it is likely that a node is connected to a node with larger degree. Therefore we expect that by continuously moving to an adjacent node with larger degree we can arrive at a kind of a root of the network.

For each node u of the network, let $N^k(u)$ denote the set of all nodes whose distance from u is at most k. The node u itself belongs to $N^k(u)$ for any $k \geq 0$. We define the parent node $\pi(u)$ of each node u as follows. The parent of u is the node whose degree is the maximum among all the nodes in $N^1(u)$. A tie is broken by taking the node with the smallest ID.

Let F be the ordered forest for G which consists of all the nodes of G and whose edges are $(u, \pi(u))$ for all nodes u of G. We say that F is defined from G by the simple rule. As the maximum out-degree of F is 1 and obviously there are no

Step0: Set the root list and the cross stack empty, then move to start node.
Step1: Walk up to the root from the current node. Check if that root exists in the root list. If yes, pop one cross-edge from the cross stack and jump to its source node, then goto Step3.
Step2: Add the current node (root) to the root list and search the tree by DFS.
Step3: Seek the next cross-edge while walking in the tree in the same way as DFS. If it is found, push it to the cross stack and go across, then goto Step1. If not, pop one cross-edge from the cross stack and jump to its source node, then goto Step3. (If the cross stack is empty when pop, it indicates that all the nodes in the connected component are visited.)

Fig. 1. The algorithm of the forest search

cycles with length more than one, F is the spanning forest, and the root of each tree in F has a maximal degree. We observe that F has the following properties by the properties of scale-free networks: As the number of large degree nodes is very small, the number of trees in F would be small, and as L_G is small, the depth of each tree in F would be small. In the following, we present the forest search to take advantage of the above observations.

The forest search walks on a given network with local topological information as if it were a forest. The idea of searching as if the search space were a forest is proposed by Avis and Fukuda as *reverse search* for hard enumeration problems [3]. An outline of searching a forest is to repeat seeking a new tree and searching it by using DFS until the target node is found. Due to the definition of the parent, it is unnecessary for DFS in a tree to memorize any tracks. We call each tree *virtual tree* because it is implicitly defined, and a graph created from the original graph by shrinking each virtual tree into a new node the *shrunken graph*.

Figure 1 presents the forest search algorithm. During this algorithm, check if the current node is the target node, and if yes, output and halt. To identify and go across trees, a *root list* to memorize the roots of the trees found so far and a *cross stack* to memorize the non-tree edges (*cross-edge*) are required. An element of the cross stack is a directed edge to go across trees, which is an ordered pair of source and destination nodes.

In effect, on a larger scale, this algorithm performs DFS also in the multigraph assuming that each tree in a forest is a node, but should visit every tree "adjacent" to the current tree because it is impossible to check whether a tree is previously found unless going to its root.

Now we present a theorem regarding as the steps in the worst case.

Theorem 1. *Let G be a finite undirected connected graph with $|V|$ nodes and $|E|$ edges, then forest search searches G in $O(|V| + |E| \cdot h)$ steps in the worst case using $O(t)$ memory, where h is the largest depth of virtual trees in G and t is the number of virtual trees.*

Proof. Traversing each tree by DFS requires $\Theta(n_i)$ steps, where n_i is the number of nodes in the tree and $\sum_i n_i = |V|$. Thus, traversing once for every tree requires

$\Theta(|V|)$ steps in total. As for the underlying forest in G, there are at most $|V|$ trees in the worst case and at most $|E|$ cross-edges, so "larger scale" DFS on multi-graph requires $O(|V|+|E|\cdot h)$ steps: The first term is the number of trees. The second term is the number of cross-edges multiplied by the largest depth h, which stems from the difference from the ordinary DFS as mentioned. As for the required space, it is obvious that only $O(t)$ space is enough for DFS on the shrunken graph. Putting these factors together, we obtain the theorem. □

We then introduce the additional rule to define a spanning forest. For each root node u in the spanning forest by the simple rule, we change the definition of the parent of u such that the parent is the node with maximum degree in $N^2(u)$. A tie is broken analogously. We can show the following:

Theorem 2. *The graph F defined by the additional rule has the following properties.*

1. *If a root node u by the simple rule has adjacent node which belongs to a different virtual tree, u is not a root by the additional rule.*
2. *F does not have a cycle with length more than one.*

Note that a cycle with length one exists for each root node.

Proof. (1) If a root node u by the simple rule is adjacent to a node v which belongs to a different virtual tree, the parent of v is not u. Therefore by the additional rule $\pi(u) = \pi(v)$ and therefore $\pi(u) \neq u$, that is, u is not a root by the additional rule. (2) From the definition of the parent, by traversing an edge from a node to its parent, either the degree increases or the ID decreases, implying that there is no cycle. □

In Section 5 we show that the number of virtual trees is dramatically reduced by the additional rule.

Note that all the walk-based local strategies mentioned in the previous sections have randomness in choosing next node, but this strategy works deterministically. We also note that, contrary to the random walk, the forest search algorithm described above assumes that the given network is static, which does not hold for real-world networks. The modification of the algorithm for adjusting to dynamic networks is easy, though we cannot guarantee the worst case complexity.

4 Theoretical Analysis

In this section we give theoretical analysis on the number of virtual trees in a scale-free network defined by the forest search. Namely, we show the following theorem:

Theorem 3. *For a scale-free network G_m^n created by the preferential attachment, the expected number of virtual trees by the simple rule is at most $c^m n$ for some constant $0 < c < 1$.*

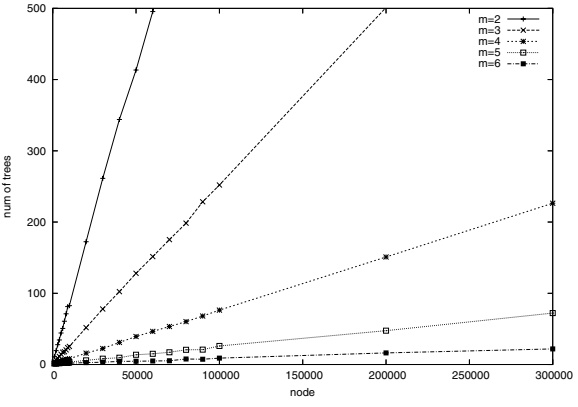


Fig. 2. The number of virtual trees in scale-free networks

Recall that G_m^n be a graph with n nodes created randomly by the preferential attachment where $m_0 = m' = m$. Then G_m^n has the same distribution as a graph generated by n-pairing [6]. Consider a partition of the set $\{1, 2, \ldots, 2mn\}$ into pairs. Sort all pairs (l_i, r_i) $(1 \leq i \leq mn, l_i < r_i)$ in ascending order of r_i's. Then the interval $(r_{m(i-1)}, r_{mi}]$ represents the node i, and each pair (l_j, r_j) represents an edge between nodes u and v where $l_j \in (r_{m(u-1)}, r_{mu}]$ and $r_j \in (r_{m(v-1)}, r_{mv}]$. We use this representation of G_m^n for the analysis.

We map the pairs (l_i, r_i) into the interval $[0, 1]$. Then r_i is a random variable and $\Pr[r_i = x]$ has a density function $2x$ $(0 < x < 1)$, and l_i is uniformly distributed on $[0, r_i]$. Let $R_1, \ldots, R_{mn}$ be the result of sorting $r_1, \ldots, r_{mn}$, $W_i = R_{mi}$, and $w_i = W_i - W_{i-1}$. The node i is represented by the interval $(W_{i-1}, W_i]$. It is shown that the expected value of W_i is $\sqrt{i/n}$, and $|W_i - \sqrt{i/n}| \leq \frac{1}{10}\sqrt{i/n}$ for all i with high probability [6].

We regard each edge (i, j) of the graph as being directed from j to i $(i < j)$. Then each node i has exactly m outgoing edges to node j's $(j < i)$, and d_i incoming edges from node k's $(k > i)$.

Lemma 1. *The expected value of* d_i *is* $\mu_i = 2m(n-i) \cdot \frac{w_i}{1+W_i}$, *and* $\Pr[d_i \geq \frac{11}{10}\mu_i] \leq \exp(-\frac{\mu_i}{207})$ *and* $\Pr[d_i \leq \frac{9}{10}\mu_i] \leq \exp(-\frac{\mu_i}{200})$.

Proof. There are $m(n-i)$ nodes with index greater than i. Consider such a node x. Then the probability that x has an outgoing edge to node i is

$$\int_{W_i}^{1} \frac{2x}{1 - W_i^2} \cdot \frac{w_i}{x} dx = \frac{2w_i}{1 + W_i}.$$

Therefore the expected value of d_i is $2m(n-i) \cdot \frac{w_i}{1+W_i}$. By the Chernoff's bound, the inequalities hold. □

Now we show the main theorem:

Proof (of Theorem 3). The number of virtual trees defined by the forest search is the same as the number of nodes whose adjacent nodes have smaller degrees

than themselves. To estimate it, we consider the probability that an adjacent node j of a node i has larger degree ($j < i$).

$$\begin{aligned}\Pr[d_j > d_i] &> \Pr\left[d_j > \frac{9}{10}\mu_j\right] \cdot \Pr\left[d_i < \frac{11}{10}\mu_i\right] \\ &> \left(1 - \exp\left(-\frac{\mu_j}{200}\right)\right) \cdot \left(1 - \exp\left(-\frac{\mu_i}{207}\right)\right)\end{aligned}$$

This probability is lower bounded by a constant for μ_i greater than some constant c'. That is, if the expected degree is greater than c', there is a constant probability $1 - c$ that the node i has an outgoing edge to a node with degree greater than that of i. If μ_i is smaller than c', the probability that $d_j > c'$ is also a constant. Therefore the expected number of root nodes of virtual trees is at most cn. Furthermore, because the node i has m outgoing edges, the probability that all adjacent nodes have smaller degrees than i is c^m. Therefore the expected number of root nodes is at most $c^m n$. □

Figure 2 shows the relation between the number of nodes and the number of virtual trees in scale-free networks for $m = 2, 3, \ldots, 6$. We can see that the number of virtual trees decreases exponentially in m.

5 Experimental Results

5.1 Number of Trees and the Size of the Largest Tree

We first see the statistics of underlying forests because the efficiency of the forest search depends on the structure of the forest which it search. We first consider the case of the simple rule. Figure 3 shows the empirical results by applying rule of choosing parent to networks of each model. There are many trees in a resulted forest in WS model (because its nodes have almost the same degree and are easy to be root nodes) and relatively much fewer in BA models, but the number of trees seems to increase linearly with the number of nodes in every model. As for the size of the largest tree in a forest, the size in BA model increases linearly, but that of other two models increases more slowly. These results suggest that in BA model, there are one giant tree and several other small trees. On the contrary in WS model, there are only small trees.

Next we show the results on the additional rule. Figure 4 shows the number of virtual trees and the size of the largest virtual tree by the additional rule for $m = 2$. We can see that the number of virtual trees is extremely small (for a network with one million nodes, the size is less than four on average), and more than 90% of nodes belong to the same tree.

5.2 Results on Finding a Path Between Two Nodes

Now we see the results of forest search. Figures 5 (without look-ahead property) and 6 (with 1-look-ahead property) show the steps by forest search plotted

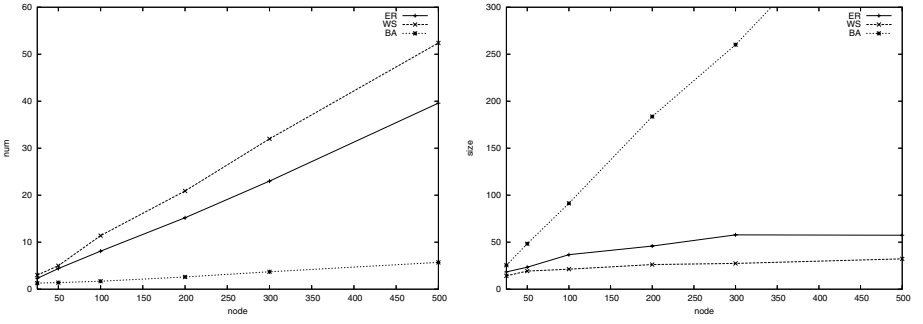


Fig. 3. The number of trees (left side) and the size of the largest tree (right side)

The figures show the results for ER, WS and BA model by the simple rule averaged over 20 distinct networks for each model

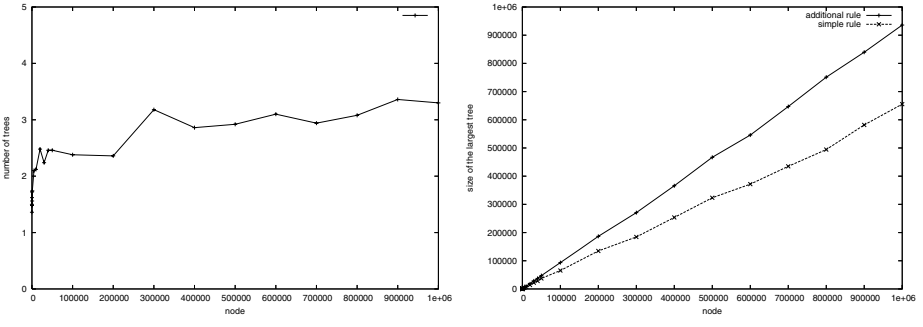
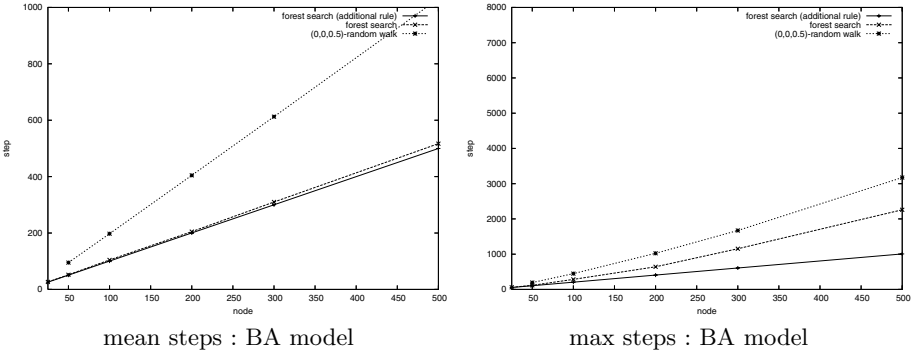


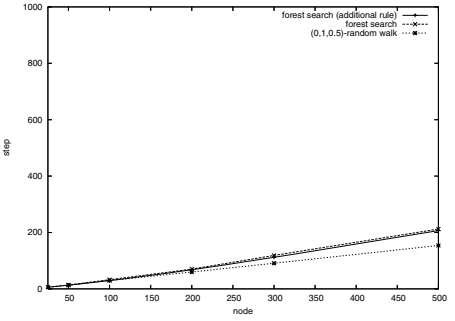
Fig. 4. The number of trees (left side) and the size of the largest tree (right side) by the additional rule



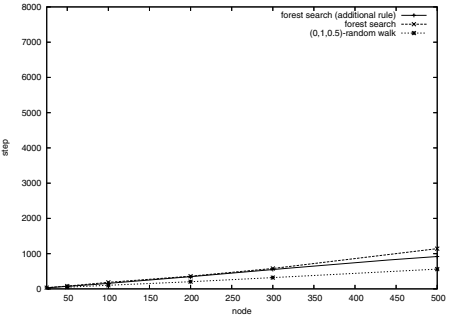
mean steps : BA model max steps : BA model

Fig. 5. Results of forest search without look-ahead property

against the number of nodes. For comparison, the result of $(0, l, 0.5)$-random walk on the same networks is also plotted with fine line in each figure. As with the random walk, the steps increase with the number of nodes.

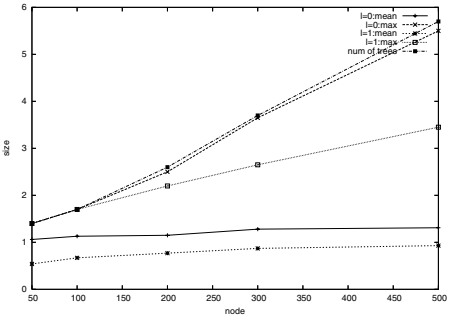


mean steps : BA model

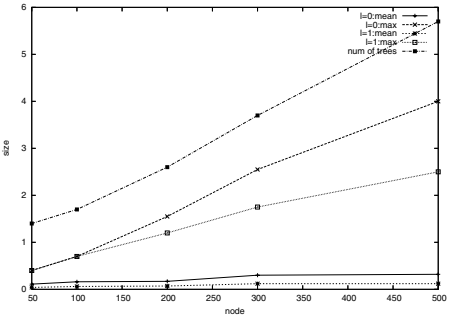


max steps : BA model

Fig. 6. Results of forest search with 1-look-ahead property



root list : BA model



cross stack : BA model

Fig. 7. Size of the root list and cross stack

On the average, for random walk without k-tabu property, the number of the candidates to one of which the agent move is 4 (average degree of the network) in a given network G, while for forest search, 1 (average number of children) in forest in G. In addition, the degrees of the adjacent nodes have little variance.

Compared with $(0, l, 0.5)$-random walk, forest search has better performance for $l = 0$ except for the max steps in WS model, but loses the precedence for $l = 1$ except for the mean steps in WS model.

As for the memory for going across trees in forest search, the size of the memory is bounded by the number of trees. Figure 7 shows the size of the memory used. These figures show the size of the root list (left side) and cross stack (right side) averaged 20 distinct networks for each model during forest search by the simple rule. Here "mean" and "max" in the legends represent the mean and max of the largest size of the list or stack in searching over all $\frac{1}{2}n(n-1)$ start-target pairs for a network, respectively. In each figure, the mean, max size for $l \in \{0, 1\}$ and the number of trees in a forest are plotted against the number of nodes.

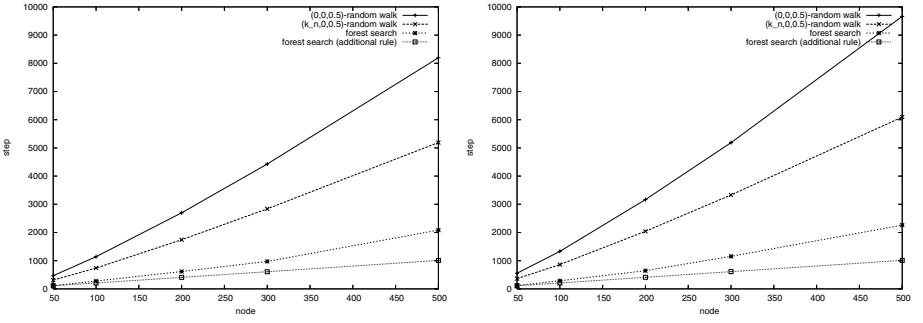


Fig. 8. Mean (left) and max (right) steps in visiting all the nodes

In every model, forest search requires almost the same space as the number of trees in the worst case. In BA model, the mean size of the memory stay almost constant against the number of nodes while those in ER and WS model linearly increase: The size of the root list is around 1 and that of the cross stack is rather larger than 0. This is also due to the giant tree in BA model.

If the additional rule is used, the size of the root list and cross stack is almost one in BA model.

In summary, forest search has the advantage for searching each tree, which is a cluster in a network in some senses and thus for searching networks in which there exists giant trees such as BA model.

5.3 Results on Traversing All the Nodes

Thus far, we take up the steps in searching the specified target node, but we consider forest search has an advantage over random walk in covering nodes of networks and we investigate the steps in visiting all the nodes of networks at least once. Figure 8 shows the result for BA model. We perform $(k, 0, 0.5)$-random walk with $k = 0$ or k_n (the doubled number of trees in underlying forest) and forest search on networks of BA model. The mean and max steps here represent the mean and max number of steps in visiting all the nodes over all n start nodes for one network, respectively. We apply each strategy to 20 distinct networks with n and calculate the average.

As for the max steps, for any graph G, forest search requires $O(|V|+|E|\cdot h)$ steps which is derived from the same analysis as in Theorem 1 while $(0, 0, 0.5)$-random walk does $O(n^2 \log n)$ [13]. Compared with both $(0, 0, 0.5)$- and $(k_n, 0, 0.5)$- random walk, forest search has better performance.

6 Concluding Remarks

We have proposed forest search: a deterministic graph exploration algorithm using local topological information of graphs. It can traverse all the nodes of

scale-free networks quickly using small memory. It is faster than a random walk strategy using the same amount of memory. We have theoretically analyzed the required memory for the algorithm. It is much smaller than the size of the graph, and exponentially decreases in the parameter m of the scale-free networks.

As future work, we give much detailed theoretical analysis on scale-free networks such as the distribution of the size of trees, the depth of the largest tree, etc. We will also consider more space efficient algorithm for exploration.

References

1. R. Albert and A.-L. Barabási, "Statistical Mechanics of Complex Networks," *Reviews Of Modern Physics*, Vol.74, 47–97, 2002.
2. L.A.N. Amaral, A. Scala, M. Barthélé and H.E. Stanley, "Classes of small-world networks," *Proceeding of The National Academy of Sciences*, Vol.97, 11149–11152, 2000.
3. D. Avis and K. Fukuda, "Reverse search for enumeration," *Discrete Applied Mathematics*, Vol 65, 21–46, 1996.
4. A.-L. Barabási, R. Albert and H. Jeong, "Mean-field theory for scale-free random networks," *Physica A*, Vol.272, 173–187, 1999.
5. B. Bollobás and O. Riordan, "The degree sequence of a scale-free random graph process," *Random Structure and Algorithms*, Vol.18, 279–290, 2001.
6. B. Bollobas and O. Riordan, "The diameter of a scale-free random graph," *Combinatorica*, Vol. 24, 5-34, 2004.
7. G. Brightwell and P. Winkler, "Maximum Hitting Time for Random Walks on Graphs," *J.Random Structures and Algorithms*, Vol.3, 263–276, 1990.
8. H. Ebel, L.-I. Mielsch and S. Bornholdt, "Scale-free topology of e-mail networks," *Pysical Review E*, Vol.66, 1–4, 2002.
9. P. Erdős and A. Rényi, "On random graphs," *Publ. Math. Debrecen*, 6:290–297, 1959.
10. U. Feige, "A tight upper bound on the cover time for random walks on graphs," *J.Random Structures and Algorithms*, Vol.6, 51–54, 1995.
11. P. Flocchini, B. Mans and N. Santoro, "On the Impact of Sense of Direction on Message Complexity," Inf. Process. Lett. 63(1): 23-31, 1997.
12. P. Fraigniaud, C. Gavoille and B. Mans, "Interval routing schemes allow broadcasting with linear message-complexity," Distributed Computing 14(4): 217-229, 2001.
13. S.Ikeda, I.Kubo and M.Yamashita, "Reducing the Hitting and the Cover Times of Random Walks on Finite Graphs by Local Topological Information," *Proceedings of The 2003 International Conference on VLSI*, 203–207, 2003.
14. Y. Kurumida, T. Ogata, H. Ono, K. Sadakane and M. Yamashita,"A Generic Search Strategy for Large Scale Real World Networks," *Proc. of the 1st International Conference on Scalable Information Systems*, ACM Digital Library, 2006.
15. P. Panaite and A. Pelc, "Impact of topographic information on graph exploration efficiency," Networks 36(2): 96-103, 2000.
16. D.J. Watts and S.H. Strogatz, "Collective dynamics of small-world networks," *Nature*, Vol.393, 440–442, 1998.

Choosing a Load Balancing Scheme for Agent-Based Digital Libraries

Georgousopoulos Christos and Omer F. Rana

Department of Computer Science, Cardiff University, P.O. Box 916,
Cardiff CF24 3XF, UK
{geolos, o.f.rana}@cs.cf.ac.uk
http://www.cs.cf.ac.uk/Digital-Library

Abstract. Digital Libraries (DLs) provide an important application area for deploying mobile agents, and with the increase in content being made available within such libraries, performance concerns are becoming significant. Current DLs often involve content servers which are geographically distributed, often necessitating information from these servers to be aggregated to answer a single query. Encoding a query as a mobile agent provides a useful means to implement such queries. However, a significant concern when this approach is adopted is the need to load balance multiple sets of such agents across the available servers. This paper focuses on an attempt to answer which load balancing scheme should be applied to an agent-based DL. A demonstration of a load balancing scheme based on the guidelines proposed in this paper with reference to Synthetic Aperture Radar Atlas (SARA) DL, consisting of multi-spectral images of the Earth, is presented. This particular DL involves processing image content, but also involves aggregation of data acquired from the images with text-based content available elsewhere.

1 Introduction

Digital Libraries (DLs) provide a useful way to group a collection of services and digital objects that may be used in a particular context. DL research has often focused on providing static content that may be subsequently accessed in a variety of ways. Recent focus on active DLs, whereby content from a collection of different repositories may be aggregated, provides useful parallels with work in Service-Oriented computing. Such repositories can be implemented using specialised hardware, and often support domain-specific interfaces. Accessing the content available within such a DL through the use of intelligent agents provides an important step in re-purposing such DL content.

The introduction of agent-technology in Digital Libraries implies the provision of agent management support within the architecture of the DL. Load balancing (LB) is one of the most important techniques that can be applied to support the management of mobile agents within an agent-based system, because apart from the even distribution of agent load among the servers, the management agents' information on LB may also be reused by other techniques that can extend the scalability of an agent-based Digital Library (ABDL).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 51 – 62, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The following section provides a brief description of different load balancing approaches, whereas the remainder of this paper focuses on an attempt to answer which load balancing scheme should be applied to an ABDL. Finally a LB scheme for an ABDL based on the guidelines proposed in this paper is demonstrated with reference to the SARA active DL.

2 Load Balancing Approaches

Generally, load balancing aims to improve the average utilisation and performance of tasks on the available servers, whilst observing particular constraints on task execution order. Assuming agents have a set of tasks to execute, it is necessary to identify how these tasks may be distributed across the available servers. Hence, workload distribution must consider both the number of agents on a server and the number of tasks being executed by each agent. Load balancing can be either static or dynamic [13] according to the multi-agent system in which it is being considered. In static load balancing tasks cannot be migrated elsewhere once they have been launched on a specified server. In dynamic load balancing a task may migrate to another server, utilising the agent's mobility.

There are two basic approaches to distribute tasks among servers: the state-based and the model-based approach. In the state-based approach, information about the system state is used to determine where to start a task. The quality of this decision depends on the amount of the state data available, and the frequency with which this state data is recorded. Gathering the data is expensive, but leads to a more accurate decision. In the model-based approach, load balancing depends on a model which predicts the system state and which may be inaccurate. Model-based approaches are more difficult to implement as they involve the derivation of an initial model, and the need to adapt the model over time.

In state-based load balancing, a common approach for managing system state and load is the *market* mechanism to value resources and achieve an efficient match of supply and demand for resources. Examples include Spawn [5] (based on a negotiated auction protocol), Dynast [14] and OCEAN [15] based on non-negotiable pricing mechanisms. System state may be accumulated in different ways, via specialised monitoring agents, such as Mats [15] and Traveler [9]. In the FLASH [20] framework a *system* agent maintains information about the whole system state and passes it to *node* agents on each server in the network. Node agents monitor locally residing mobile agents. *User* agents (which are mobile) are responsible for the load balancing of the parallel application, and migrate through a cluster searching for free resources. Their migration decisions are based on internal states as well as internal and external events.

Almost all the systems that explore the model-based approach use distribution of CPU load and expected process/task lifetime to decide if and when to migrate. Malone's Enterprise [18] system uses a market mechanism, and the Challenger [1] system uses a learning-based approach. Eager et al. [10] utilise concurrent execution to improve resource usage. Most of these approaches however cannot easily adapt to changing system workloads.

Finally, SARA [6] utilises a LB scheme based on a combination of the state-based and model-based approach of LB. The state-based part of LB is similar to FLASH, apart from the fact that decisions on load balancing are supported though the stationary management agents and not the mobile agents (see section 3.3); whereas the model-based part of LB is described in [7]. Note that SARA was also the first system to employ the FIPA-compliant gateways [8] to enable FIPA interoperability with external FIPA-compliant agent-based systems. The proposed approach is therefore also suitable for other systems that utilise the FIPA architecture (based on the Agent Management System (AMS), the Directory Facilitator (DF) and a limited set of FIPA performatives).

3 Choosing the Appropriate Load Balancing Scheme for an ABDL

Research experiments [4] prove that dynamic LB outperforms the static placement scheme by 30-40%, consequently the focus on choosing an efficient load balance technique an ABDL is based around dynamic LB.

The objective of market-based approaches on LB is to value resources and achieve an efficient match of supply and demand for resources. This may be achieved by using only a price, match offers and bids, or by employing more sophisticated auction protocols. Therefore LB in this case is directly related and influenced by the amount of currency the agents have. The higher is the currency possessed by an agent, the more advantageous it becomes in utilising server resources. Even in the Vickrey auction (in where the price paid by the winner agent of the auction equals the second-highest bid placed), agents with less currency have limited chances of winning an auction i.e. utilising resources for the execution of their tasks. Consequently, market-based approaches tend to be priority-based. The aim of LB in an ABDL is to evenly distribute agents among the servers, as well as to equitably serve them. The agents' tasks are carried out simultaneously and there are no priorities between the agents; agent task completion times therefore do not necessary imply a higher priority. Since, the objective in an ABDL is to serve equitably the agents without any priority levels, market-based approaches are impractical for such architectures.

As it has been mentioned in section 2, the state-based approach of LB is based on the information about the system state, which is used to decide the server where a task must be started. Consequently, the nature of information acquired impacts the effectiveness of the LB technique. In addition, in distributed systems where network and server conditions change dynamically, for LB to be effective, it should adapt quickly to those changes. LB approaches which use mobile agents to roam through the network searching for free available resources, lack this kind of adaptiveness. In such approaches agents have to migrate from server to server until they find the needed resources, which is likely to result in network load and in the increase of servers' utilisation. This is because multiple agents simultaneously migrate thought the network and since they are active they consume resources e.g. memory. These agents only have information regarding the servers they have visited, but during their itinerary a lot of changes might take place on the previously visited servers of which the agents will be unaware of. For instance, during an agent's itinerary, resources on a

server that has already been visited might become available, but the agent will keep on migrating because it is impossible for it to be informed about this change.

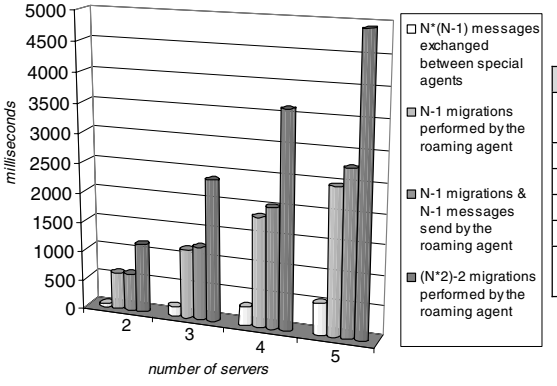
3.1 Gathering, Distributing and Updating System State Information

The ability of an agent to have knowledge of the overall system state can be achieved with the introduction of special agents on every server which monitor the local server resources, and exchange their information between themselves. This overcomes the adaptiveness problem, optimises the load balance decisions based on the overall system state information and decreases the network load by eliminating unnecessary agent migrations. A message is faster to transmit in contrast with the time an agent needs to be serialised and migrated. Therefore, where there is a sole roaming agent to gather the overall system state in a network of *N* servers, such an agent has to be serialised and do *N-1* migrations, while the special agents have to exchange $N*(N-1)$ messages between themselves to achieve the same result. However, the roaming agent, after it has finished its itinerary, has to either migrate back to all the previously visited servers (i.e. do additional *N-1* migrations) or send each of them a message (i.e. total of *N-1* messages have to be transmitted) containing information on the local system states that have been collected so that every server is aware of the overall system state.

An experiment on a 100Mbit/s Fast Ethernet network of five servers was conducted to compare the approach of using a roaming agent in contrast with the existence of special agents on each server for gathering and distributing the overall system state information between the servers. The servers used were Intel Pentium 4 of similar CPU processing powers ranging from 1.8 to 2.1 GHz running Microsoft Windows XP utilising the Voyager [19] agent platform. The experiment was conducted on unloaded servers with virtual local system state information ranging from 150 to 200 bytes each, consisted of information about the server's utilisation, number of virtually active agents, and availability of resources. The initial size of the roaming agent was 2.8Kbytes with the functionality of migration and storing local system state information within. Note that the size of the roaming agent was increasing on each migration due to update in state information that had to be kept. The time needed for a single message (containing local system state information of a server) to be transmitted was 21-36ms, the time of agent serialisation was 31-47ms, whereas the time required by the agent to migrate itself and store the local server system state information was 564-678ms. The time needed to create a reference to a proxy (i.e. special agent) was 93-125ms, but it has not been considered in the evaluation since it is only needed once during the lifetime of the special agents.

The number of servers in the chart of Figure 1 starts from 2, since a remote agent migration requires the existence of at least two servers. The 1st and 2nd bar of each graph in Figure 1 represents the time needed from both approaches to gather the initial overall system state, where the 3rd and 4th bar includes the additional time (of the roaming agent approach) to distribute information to all the servers. As can be seen the special agents approach outperforms the roaming agent one. However, in order to observe the behavior of both approaches in a network of tens of servers, because it was quite difficult to run the same experiment for more than five servers (due to lack of availability in computing facilities), based on the information provided in the table

of Figure 1 it has been calculated that for *N* equals to 44, the time required in the special agents approach to exchange $N*(N-1)$ messages is greater than *N–1* migrations and *N–1* message exchange (performed in the roaming agent approach).



Parameter	*Value*
Local system state information	150-200 bytes
Size of roaming agent	2.8 Kbytes
Message transmission	21-36 ms
Agent serialisation	31-47 ms
Agent migration	564-678 ms
Creation of reference to a proxy	93-125 ms

Fig. 1. Comparison of roaming versus special agent

This is due to the fact that every message exchanged between the special agents containing local system state information of a server is approximately the same in size, whereas in the roaming agent approach each message sent to a server differs in size. Although the time required in the special agents approach to transmit $N*(N-1)$ messages corresponds to the same number of local system state information that have to be exchanged, in the roaming agent approach the time required to send *N–1* messages corresponds to the transmission of $N \cdot (N-1)/2$ local system state information. This is because during the roaming agent's migration, every visiting server is informed by the agent concerning the local system state of previously visited ones, but already visited servers do not have information on those that have not been visited yet. In this instance, the message sent from the roaming agent to the first server of its itinerary contains information on the local system state of every visited server, whereas the message sent to the server before the last one (in the agent's itinerary) contains information only for the last visited server. Therefore, in a network of more than 43 servers roaming agent approach turns out to be more efficient.

Nevertheless, in dynamic environments where changes frequently take place the special agents approach is preferable in spite of the number of servers. This is because in the event of a server system status changing the rest of the servers can be informed with the cost of exchanging *N-1* messages, while in the alternative approach, the roaming agent has to perform its task from the beginning i.e. in the best case, do *N-1* migrations and exchange *N-1* messages (of greater size in comparison with those exchanged in the special agents approach). Of course, the ideal approach in a network comprised of more than 43 servers, would be to initially gather the system state information using a roaming agent but keeping it updated with the use of special agents placed on each server.

3.2 Special Agents in the State-Based Load Balance

Although the approach of using stationary monitoring agents in contrast to roaming agents for gathering and disseminating overall system information is preferable, different policies exist in relation to the stationary agents' perspective of the system. Policies range from Direct-Neighbor policy (i.e. every special agent communicates only with its direct-neighbor stationary agent and exchanges local system state information only with them; and load balancing actions are limited to two direct-neighbor servers) to All-Neighbor policy, where all stationary agents exchange local system state information between themselves.

Research experiments show that policies where the special agents' perspective of the system is limited suit well highly dynamic applications. In "slowly dynamic"[1] applications[2], the wider the special agents' perspective is, the better load balance quality can be achieved; where the total number of migrations is also diminished. Zambonelli[11] introduced a new scheme of information exchange in neighboring load balance policies, in which the system state (load) information transmitted is distorted to enable special agents take into account a wider perspective of the system and overcome the limit of the local view. This is achieved by weighting the load of a server with the average load of its neighbour servers. Though, his experiments show that the transmission of distorted load information provides high efficiency unless the dynamicity of the load becomes too high i.e. near to 70%, in which case it is preferable to exploit non-distorted load information.

A different approach of achieving efficient load balancing in neighboring load balancing policies with respect to the global view of a system was followed by Keren and Barak [4], with the ability of the stationary monitoring agent to dynamically change their neighbors i.e. their perspective of the system. In their framework for parallel computing, each server's utilisation (load index), which is the only system state information shared between the stationary agents, is exchanged using two simultaneous dissemination schemes. First, each server - represented by a stationary agent - sends its load index by attaching it to messages sent by its local agents to other servers. Load indices are also sent to randomly chosen servers using a probabilistic load exchange algorithm. The net result is that for each time unit, every stationary agent has information about a subset of other agents. The load balance migration decisions are conducted by the stationary agents based on an algorithm which is executed periodically in an asynchronous manner. The algorithm determines the performance gain in migrating some of its agents to other servers, which is a function of the resulting change in the load and the inter-server communication, and if a substantial gain is obtained the migration follows. The migration decision algorithm is also executed by every stationary agent of the same subset.

Despite the fact that in dynamic environments the narrower the perspective of stationary agents (i.e. information only about neighboring servers), the better the load balancing that can be achieved, in an ABDL architecture the stationary agents must have a global view of the system. This is because an active Digital Library may be composed of a collection of different information and computation resource servers (though some might be replicated). Since the agents' tasks are resource-depended and

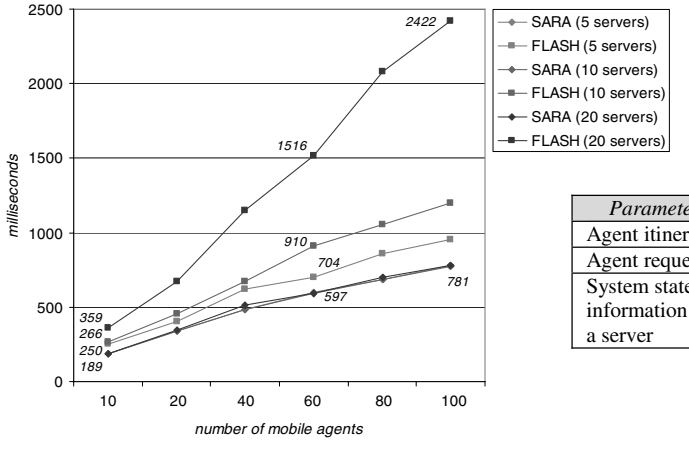
[1] According to [2] these are systems that do not change frequently.

the resources needed by each task are unknown before its initiation, efficient load balancing can only be achieved with a global view of the system, provided by stationary agents.

3.3 Providing Load Balancing Decisions Via Stationary Agents

The architecture of the FLASH framework utilises stationary agents with a global view of the system for gathering, disseminating and updating the system state information. The SARA system uses a similar concept. While in FLASH load balancing decisions are supported by the mobile agents based on their intelligence and the global system state information supplied to them by the stationary agents, in SARA the control over LB decisions is on the stationary *management* agents. Therefore, in SARA the management agents record system state and optimise the load of the available servers. Although the mobile agents may be programmed with the intelligence to give priority to the overall system optimisation and not to their own tasks, giving management agents the control over the load balancing decisions leads to the following benefits:

i) minimisation of information transmitted: The management agents balance the load of mobile agents among the servers by defining their itinerary. Once a mobile agent is created, it communicates with its local management agent, gives its requirements i.e. specifies its task, and waits for a response. The management agent in return, based on the agent's requirements and the current system state information constructs the mobile agent's itinerary and sends it back to that management agent. Consequently, only two messages are exchanged between a mobile agent and a management agent: the agent's requirements and the agent's itinerary. In the case where the mobile agent would be in control of the load balancing decision, every mobile agent would have to retrieve from a management agent the overall system state information in order to make a reliable decision; which results in unnecessary duplication of information.



Parameter	*Value*
Agent itinerary	15 bytes/server
Agent request	60-80 bytes
System state information of a server	700-750 bytes

Fig. 2. Interaction between the special/management agent and the mobile agents

Figure 2 shows the time spent (in milliseconds) on the interaction of a management agent with a number of mobile agents on a single server, according to the amount of data that has to be exchanged. The experiment was conducted on an Intel Pentium 4 1.8Ghz server running the Java-based Voyager agent platform on Microsoft Windows XP, where an agent's itinerary needed 15 bytes per server, its request was approximately 60-80 bytes and the system state information of a server encoded in XML at 700-750 bytes. Although the difference in time between the two approaches is minimal i.e. a few seconds, as the number of servers on the network is increased considering the total time of the agents' interaction from each server, this difference becomes important. Finally, from the above chart it can be observed that in SARA the agents' interaction time, irrespectively of the variable introduction of participants is almost uninfluenced by the number of servers (from 5 to 20) employed in the network.

ii) minimisation of the mobile agent's size: Decisions on load balancing are based on an algorithm (a model) that accepts as input an agent's requirements and the system state information, and gives as output an itinerary of servers where the particular agent should migrate to. In SARA, the management agents provide this functionality. Alternatively, every mobile agent must have this decision support algorithm within itself. One of the most important characteristics of a mobile agent is its size; the smaller the mobile agent is in size, the faster it can move through the network. Hence, by giving the management agents the control over LB decisions, the size of the mobile agents is preserved to its original size.

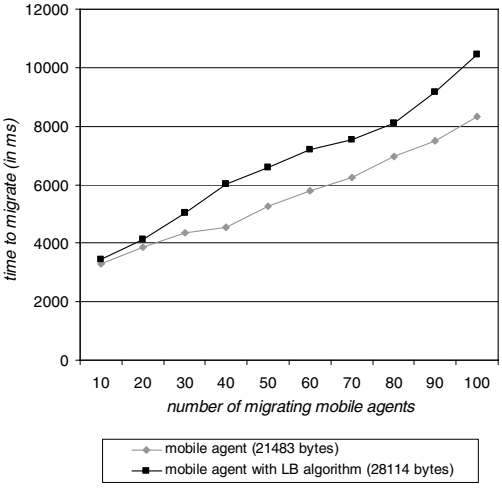


Fig. 3. Migration times of variable number of migrating agents

Figure 3 shows the influence of the mobile agent's size on its migration. The experiment was conducted on a 100Mbit/s Ethernet network with two Sun Ultra 5 Workstation of a 270 MHz UltraSPARC-IIi 64-bit processor running on Solaris 8, utilising the Voyager agent platform. In the experiment two types of mobile agents with different sizes were used. One was the mobile User Request Agent used in SARA of 21483 bytes, and the second was the same agent extended with the load

balancing decision algorithm, resulting in an agent of 28114 bytes. The total migration time increases with the number of concurrently migrating agents, and the larger a mobile agent is in size the more time is required for its migration when the number of concurrent migrating agents is increased.

iii) system optimization: Management agents may also maintain a record about mobile agents that are active on their host platform. This information may include the task of the agent, the resources that have been used, the time taken to complete the task and the site where the results of the task have been stored. This information is used to support load balancing, and for undertaking similarity analysis between agent requests. Hence, if an agent's task (request) is identical to a task already performed, the task does not have to be repeated and previous results can be retrieved. If an agent's task is similar but not exactly the same as an already accomplished task, the model applied could be able to determine if it is worthwhile for the agent to process the results of the existing task or to re-execute the task.

Management agents therefore contribute to a mobile agent's migration optimisation by defining the itinerary for an agent according to its task and the current system state information. For instance, an agent with a task of acquiring a collection of images and filtering them on a compute server against a user's custom analysis algorithm will be guided by a management agent. *How* the agent migrates (i.e. just the agent itself, the agent containing the custom algorithm etc.) and *when* it should load any classes necessary for the accomplishment of its task is its own choice based on its rules and the status of its task. Even in the event of a server failure, the mobile agent is capable of moving autonomously to the next available server of its itinerary, communicating with the local stationary agents without being controlled by the management agents.

4 SARA Active DL

SARA is an active DL of multi-spectral remote sensing images of the Earth from the SIR-C Shuttle mission. Web-based online access is provided to a library of data objects at Caltech and the San Diego Supercomputer Center in the US, and the University of Lecce in Italy. A prototype multi-agent system, which comprises both intelligent and mobile agents, has been developed to manage and analyse data in the SARA DL [12].

The SARA architecture is composed of a collection of information and Web servers, each of them supporting a group of agents. Information servers support Local Interface Agents (LIA), whereas Web-servers support User Interface Agents (UIA). The information-servers manage the computational resources and data repositories to support the SARA active DL – where the data repositories generally contain pre-processed images or geospatial data about a given region. Figure 4 represents the SARA architecture and the multi-agent interaction.

The architecture's approach is based on localizing the most complex functionality in non-mobile LIAs, which remain at one location, provide resources and facilities to lightweight mobile agents that require less processor time to be serialised, and are therefore quicker to transmit. LIAs are stationary agents that provide a set of pre-defined services. The primary motivation for using mobile agents are: (a) the avoidance of large data transfers – of the order of Terabytes, consisting of sometimes proprietary data, (b) the ability to transfer user developed analysis algorithms, and (c) the ability to utilise specialised parallel libraries.

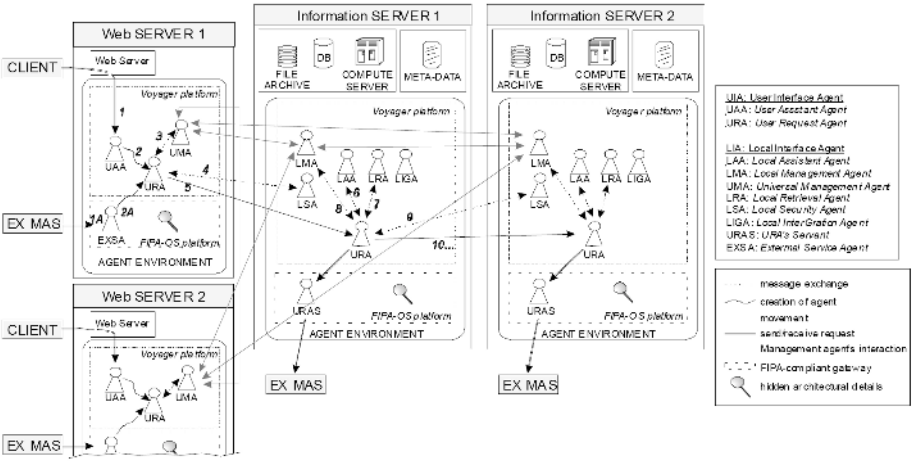


Fig. 4. The SARA agent-based architecture

The front-end interface allows a user to query a collection of SAR images, and provides the ability to further fuse the results with data available locally to the user. Such a request is received from UAA – trace the numbers in figure 4 – that creates a mobile agent i.e. URA on behalf of the user and forwards the request. After dispatch, the mobile agent is responsible of migrating to the information-servers and interacting with the local stationary agents to fulfill the user's request. The stationary agent LAA is responsible for supplying the URA with information about accessing the data repository at the local server, whereas the LRA's objective is to execute a query on the data source on behalf of the URA.

4.1 The SARA LB Scheme

The SARA load balancing scheme is a combination of the state-based and model-based approach of LB. In SARA architecture a management agent is assigned to every server (see Figure 4) in order to monitor the local resources. On initialisation every management agent gathers local system state information and exchanges it with other management agents using multicast messaging. Subsequently, only updated information is exchanged between themselves. In this instance, the local system state of each server forms the overall system state information (a global view of the system) that every management agent maintains. Load balancing of the URA mobile agents that encapsulate users' queries is supported by the management agents based on a model - which accepts as input an agent's requirements and the system state information, and gives as output the appropriate server(s) where the particular agent should migrate to in order to fulfill its task.

The model is a function of the: (1) agents' tasks, (2) servers' utilisation (workload), (3) availability of resources at the server, (4) network efficiency, and adapts over time due to the information gathered from the management agents on system state. The utilisation of a server is expressed by Malone's [18] formula $U = (\alpha \cdot \mu)/L$; where, α corresponds to the number of agents on that server (assuming that there is a one task

per agent), μ to the average task time of the α agents, and L to the total processing power of the hosting server. A server's utilisation in relation to its processing power L can be estimated. Such a comparison on utilisation values helps identify a server that will complete first. Accuracy of estimating a server's utilisation is based on a perfect estimation of the agent task lifetimes. The more accurate the average task time μ of agents α, the more reliable the corresponding server's utilisation.

The actual utilisation of a server can be measured by using specialised routines/utilities (like *xload* or *ps,* available on the Unix operating system) that provide the CPU usage. The difference between these routines and Malone's approach is that while the former provides the current utilisation (CPU usage) of a server the latter also denotes a value of when a server will be unloaded, and may be used as a predictive tool. As a server's CPU usage changes frequently, decisions on load balancing should not only rely on the current utilisation of each server, but rather on which server will be unloaded first in the future. The advantage of using Malone's formula is that apart from estimating utilisation, it is also possible to predict it before the assignment of a new task to that server, given that the lifetime of the corresponding task is known.

The SARA model may be easily amended to be employed by other systems utilising active archives. In fact apart from the predictive nature of the model, in systems where the lifetime of tasks cannot be estimated or tend to be erroneous, system developers can take advantage of the adaptability of the model to cater for variable system workloads. The adaptability of the model provides a means of self-adapting to such error estimations by monitoring the utilisation of every server and amends it when it is miscalculated. Further details on SARA LB scheme can be found in [12].

5 Conclusion

An approach that compares the management of state-based information to facilitate load balancing decisions in a mobile-agent system is presented. The comparison relates the distribution of load information using mobile agents vs. messaging between stationary management agents. The approach is particular suited to Digital Libraries, where content integration may be necessary across a number of servers (such as in the SARA Digital Library). It is found that stationary agent based approaches are useful when utilising large mobile agent sizes (due to serialisation overheads).

References

1. A. Chavez, A. Moukas, P. Maes, "Challenger: A multi-agent system for distributed resource allocation", in proceedings of the 1st Int. Conference on Autonomous Agents, ACM Press, Marina del Ray, CA, USA, 1997.
2. A. Corradi, L. Leonardi, F. Zambonelli, "On the effectiveness of different diffusive load balancing policies in dynamic applications", conference on High-Performance Computing and Networking (HPCN-98), LNCS, No. 1401, Springer-Verlag (D), April 1998.
3. A. Kambil, "Different auction models", http://pages.stern. nyu.edu/~akambil/teaching/cases/auction/appendix.html.

4. A. Keren, A. Barak, “Adaptive placement of parallel java agents in a scalable computing cluster”, in proceedings of the Workshop on Java for High Performance Network Computing, ACM Press, Palo Alto, CA, USA, 1998.
5. C.A. Waldspurger, T. Hogg, B.A. Huberman, J.O. Kephart, W.S. Stornetta, “Spawn: a distributed computational economy”, transactions on Software Engineering, 18(2):103 117, 1992.
6. C. Georgousopoulos, O.F. Rana, “Combining state and model-based approaches for mobile agent load balancing”, in proceedings of Symp. on Applied Computing (SAC03), ACM press, ISBN 1-58113-624-2, held in Melbourne, Florida, USA, 2003, pp. 878-885.
7. C. Georgousopoulos, O.F. Rana, “Performance-sensitive Service Provision in Active Digi-tal Libraries”, in proceedings of Int. Conference on e-Technology, e-Commerce and e-Services (IEEE05), IEEE Computer Society press, ISBN 0-7695-2274-2, held in Hong Kong, 2005.
8. C. Georgousopoulos, O.F. Rana, A. Karageorgos, “Supporting FIPA interoperability for legacy multi-agent systems”, in proceedings of 4th Agent Oriented Software Engineering (AOSE) workshop of AAMAS’03 conference, Springer-Verlag LNCS 2004, ISBN 3-540-20826-7, held in Melbourne, Australia, Sydney, 2003, pp. 167-184.
9. C.Z. Xu, B. Wims, “Traveler: a mobile agent infrastructure for wide area parallel computing”, in proceedings of the IEEE Joint Sump. of 1st Int. Symp. on Agent Systems and Applications and 3rd Int. Symp. on Mobile Agents, Palm Springs, 1999.
10. D.L., Eager, E.D. Lazowska, J. Zahorjan, “Adaptive load sharing in homogeneous distributed systems”, IEEE Trans. on Software Engineering,vol SE-12,1986,pp.662-675.
11. F. Zambonelli, “How to improve local load balancing policies by distorting load information”, in proceedings of the 5th International Conference on High-Performance Computing, IEEE CS Press, Madras (IN), December 1998.
12. SARA active DL,http://www.cs.cf.ac.uk/Digital-Library.
13. J. Gomoluch, M. Schroeder, “Information agents on the move: A survey on load-balancing with mobile agents”, in Software Focus, vol. 2, no. 2, Wiley, 2001.
14. M. Backschat, A. Pfaffinger, C. Zenger, “Economic-based dynamic load distribution in large workstation networks”, in proceedings of the 2nd Int. Euro-Par Conference, volume 2, Lyon, France, 1996, pp. 631-634.
15. R. Ghanea-Hercock, J.C. Collis, D.T. Ndumu, “Co-operating mobile agents for distributed parallel processing”, in proceedings of the 3rd Int. Conference on Autonomous Agents (AA99), ACM press, Mineapolis, USA, 1999.
16. OCEAN, http://www.cise.ufl.edu/ ~mpf/ocea n/index.htm.
17. R. Ghanea-Hercock, J.C. Collis, D.T. Ndumu, “Co-operating mobile agents for distributed parallel processing”, in proceedings of the 3rd Int. Conference on Autonomous Agents, ACM press, Mineapolis, USA, 1999.
18. T. Sandholm, “Distributed rational decision making”, in Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence, Weiss, G. (ed.), MIT press, 1999.
19. T.W. Malone, R.E. Fikes, K.R. Grant, M.T. Howard, “Enterprise: A market-like Task Scheduler for Distributed Computing Environments”, in: The Ecology of Computation. Ed. Huberman, B. A. Elsevier, Holland, 1988.
20. Voyager agent-platform, Recursion Software Inc., http://www.recursionsw.com/osi.asp
21. W. Obeloeer, C. Grewe, “Load management with mobile agents”, in proceedings of the 24th EUROMICRO Conference, IEEE, 1998, pp. 1005-1012.

A Novel Software Solution for Localized Thermal Problems

Sung Woo Chung[1,*] and Kevin Skadron[2]

[1] Division of Computer and Communication Engineering,
Korea University, Seoul 136-713, Korea
swchung@korea.ac.kr
[2] Department of Computer Science,
University of Virginia, Charlottesville 22904, USA
skadron@cs.virginia.edu

Abstract. In this paper, we propose a temperature-aware DFS (Dynamic Frequency Scaling) technique using the performance counters that is already embedded in the commercial microprocessors. By using performance counters and simple regression analysis, we can predict the localized temperature and efficiently schedule the tasks considering the temperature. The proposed technique is especially beneficial to potential localized thermal problems that are inevitable due to limited number of costly CMOS thermal sensors. When localized thermal problems that were not detected by thermal sensors are found after fabrication, the thermal problems can be avoided by the proposed software solution without re-fabrication costs. The evaluation results show that the proposed technique is comparable to the DFS technique using CMOS thermal sensors.

1 Introduction

Reducing energy consumption has been one of the most interesting research topics in the computer architecture field. As technology trends leads to packing transistors ever more tightly, power densities are increasing rapidly. The higher heat flux leads to higher cooling costs-otherwise high temperature might cause the unexpected functional errors or permanent damage of microprocessors, especially in high-performance microprocessors. Thus, it is important to control the temperature as well as the energy consumption. To control the temperature, a couple of techniques have been proposed. One is to use the cooling fan to lower the temperature of a chip and the other is to make a heat spreader more efficiently. For example, Intel's Pentium 4 already has a cooling fan and an efficient heat spreader [20][24] and PowerMac G5 has huge cooling pumps [18]. To solve the thermal problems, on the computer architectural level, pipeline throttling, DVS (Dynamic Voltage Scaling), and DFS (Dynamic Frequency Scaling) have been proposed [2][14][16].

To control the temperature, we need to know the actual temperature of the functional block that needs to be controlled. In the Pentium 4, there are two independent thermal sensors [19]. By using on-die temperature sensing circuit and a fast acting temperature control circuit, the processor can rapidly initiate thermal management

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 63–74, 2006.
© Springer-Verlag Berlin Heidelberg 2006

control. The Pentium 4, however, only uses one of its sensors for thermal management; the other is for external use and is not located near any anticipated hotspots. In fact, hotspots may move over time, depending on which on-chip functional blocks (register fie, integer arithmetic, floating-point arithmetic, etc.) are most heavily used [8]. As technology scales down, power density increases which might lead to more localized hotspots. Temperature differences become exponentially larger with distance, so a single thermal sensor does not cover a large chip like the Pentium 4. In future high-performance microprocessors, more than ten thermal sensors are expected to be embedded in a microprocessor. However, the number of thermal sensors is limited, because they are too expensive to be placed in all the potential hotspots. When potential hotspots that do not have thermal sensors are found serious after fabrication, it is impossible to resolve the localized thermal problems without re-fabrication, using previous techniques.

We chose DFS instead of DVS for the scheduling policy. There are three reasons. 1) The frequency transition at the high Vcc is done within few microseconds, which takes much less, compared to the voltage transition [11]. 2) We found a linear proportional relation between the frequency and the temperature by using simple regression analysis. On the other hand, the voltage is not linearly proportional to the temperature, which makes it difficult to find a relation between them. 3) In terms of reliability, the supply voltage scaling reaches a plateau, since the difference between supply voltage and threshold voltage should be kept large enough [6]. Thus, this paper proposes a ***DFS technique using performance counters*** that efficiently controls the temperature of the localized hotspots. The localized thermal problems that were found after fabrication can be resolved by using the proposed technique.

The rest of this paper is organized as follows. Section 2 presents related works. Section 3 explains the temperature-aware DFS scheduling using performance counters. Section 4 describes the experiment methodology and Section 5 shows the efficiency of the proposed technique. Section 6 concludes the paper and describes some avenues for future works.

2 Related Works

Huang et al. [4] proposed a DVS-based technique for thermal control. Though they investigated the memory hierarchy, they did not examine other hot functional blocks such as register files. Brooks et al. [2] set a constant threshold power and they applied five thermal control techniques (clock frequency scaling, voltage and frequency scaling, decode throttling, speculation control, and I-cache toggling), when the threshold power was exceeded. They found DFS and DVS to be inefficient because of the invocation overhead. However, the inefficiency may be due to the short sampling period (10K cycles) and large invocation overhead (more than 10 ms). Skadron et al. [12] proposed formal control theory for dynamic thermal management. The previous studies used constant trigger temperature (or power) and fixed response. In contrast, they allow the fetch-toggling rate to be changed according to the thermal history that may need additional storage. There are some previous works [8][10] on thermal management in SMP systems, which schedules the tasks making use of the idle SMP nodes. Srinivasan et al. proposed the predictive dynamic thermal management by profiling

multimedia applications [16]. Most of these researches are based on the thermal sensors to measure the temperature.

Though the number of thermal sensors is limited by design budget, localized hotspots are too serious to be ignored [8]. Alternative to the thermal sensor is the performance counter that was already embedded in microprocessors to evaluate the performance. There have been several studies on using performance counters. Brooks et al. proposed using performance counters to find activity factors [2], where details were not proposed. Bellosa et al. proposed formulas that correlate the activity factor to energy that is eventually correlated to temperature [1]. They tried to manage the temperature by controlling power consumption [1][16] . They only concentrated on the overall temperature (not on the localized hotspots). Lee et al. [6] also proposed runtime temperature sensing using performance counters, which is accurate but incurs some computational complexity, because they use full HotSpot [13][14].

In this paper, we present a software technique using performance counters that can investigate the localized hotspots. To estimate the temperature of functional blocks, we only have to calculate a simple linear formula with inputs from the activity factor (the number of accesses) of the functional block. The linear formula is established by simple regression analysis. The data (activity factor(X) and temperature(Y)) for regression analysis can be obtained from real measurement in laboratories or from accurate simulations. In this paper, the parameters for regression analysis are obtained from simulation using HotSpot [13][14]. Though the performance counters are read every 10 ms, the estimated temperature was shown to be accurate enough [3]. In addition, the frequency transition overhead that is done every 10 ms is negligible [11].

3 Temperature-Aware DFS Technique Using Performance Counters

We examine two methods to measure the temperature: One is using CMOS thermal sensors and the other is using performance counters. The former is more accurate but needs CMOS thermal sensors. In other words, the thermal sensors should be placed in the localized hotspots before fabrication. The latter is less accurate but does not need additional hardware, since performance counters are already embedded in commercial microprocessors. On-chip sensors are now widely used to measure the temperature but are believed by many designers to be too expensive to be placed in all the potential localized hotspots. To alleviate the cost of the thermal sensors, only very probable localized hotspots have the thermal sensors. After fabrication, there is a possibility that severe localized hotspots that were not detected at the time of validation, are found. For this case, we propose a temperature-aware DFS technique using performance counters for sensing the temperature of the possible localized hotspots. Originally, the performance counters are used to count specific micro-architectural events for debugging and performance measurements. However, we can examine lots of localized hotspots by utilizing performance counters. For example, in the Intel Pentium 4, there are 45 configurable events and 18 physical performance counters, which implies that we can estimate temperatures of the 45 functional blocks in the microprocessors [15][27] .

For the temperature-aware scheduling, simple *offline* regression analysis [3] is used to find a simple relation between selected values of activity factor and observed values of temperature. Please recall that the most probable value of Y can be predicted for any value of X by simple regression analysis. Temperature can be estimated using a simple formula ($T=ax + b$, where T is temperature, X is activity factor, and a and b are coefficients). We only have to consider only the activity factor of the functional unit that is investigated. The key observation is that the regression captures second-order contributions from other functional units. We did try multiple regression analysis with the current activity factor and the previous activity factor. Results were at best minimally improved compared to results from simple regression analysis, and in fact the accuracy with multiple regression analysis was sometimes worse.

At runtime, multiplying the activity factor by the regression coefficient is required for temperature measurement. Although it is feasible to re-compute temperature every cycle, this is wasteful, since even at the fine granularity of architectural units, it takes at least 100K cycles until the temperature rise by 0.1C [14]. We chose a sample period 10 ms, which is the scheduling granularity of commercial operating systems and creates a natural opportunity for software to read the performance counts. For our CPU clock rate of 2.6 GHz, this works out to be sampling period of 26 M cycles. This is in any case the minimum granularity at which software techniques could perform any kinds of thermal management. For example, to compute the temperatures of the integer register file, we only utilized the IIPC (Integer Instructions Per Cycle) statistic. Although the peak temperature estimation error was small, there were times when our technique under- or over-estimated temperatures by as much as 10 degree. These large differences only occurred when the performance counter technique responded faster than the actual temperature. The reason is that the proposed technique is linearly proportional to the IIPC so that the estimated temperature changes quickly, whereas the actual temperature changes gradually. We did not mediate these spikes and dips, since we may be able to schedule tasks more efficiently if we know the temperature tendency (increase/decrease) in advance.

In this paper, we compare the scheduling efficiency using the thermal sensors to that using the performance counters. In the conventional technique using thermal sensors, the frequency is lowered when the temperature is more than (or same as) the threshold temperature and the actual temperature is measured from the thermal sensors. On the contrary, in case of the proposed technique using performance counters, the temperature is estimated from the activity factor so that the frequency is lowered when the activity factor (instead of the actual temperature) is more than a threshold.

4 Experiment Methodology

The processor used for the experiments is a 2.6 GHz Pentium 4, 130 nm Northwood core. The typical power dissipation is 69.0 W, and the operating voltage is 1.6 V [23]. The processor supports hyper-threading technology, which allows the processor to run two threads simultaneously. This means that the task that regularly reads the performance counters and calculates the temperature interferes minimally with user tasks: not

only does it consist of only a few instructions, but hyper-threading fits these few instructions into empty execution slots as instructions are issued within the processor.

The performance counters are used to count specific micro-architectural events for debugging and performance measurements [21]. Each counter is associated with one counter configurable control register (CCCR), which determines the specific counting scheme. The event selection control registers (ESCRs) determine which event is to be counted. A simplified device driver, adapted from the abyss device driver [27], is used to configure all the control registers and read the performance counters.

The temperature model requires the geometric specifications and the floorplan layout of the processor. We derived the configurations of Pentium 4 to configure HotSpot [13][14]. These parameters are based on design schematics found in [23]. We also use the floorplan layout that was adapted from the Northwood core die photo [22].

Though we are able to investigate the temperature of 45 functional blocks through performance counters, we concentrate on the register file which is known as one of the hottest functional blocks. In the simple regression analysis, IIPC is X (selected value) and the temperature is Y (observed value). The actual temperature is obtained from the HotSpot [13][14] that was proven to be accurate. To use the performance counters, the Hotspot was modified to be based on a model by Isci and Martonosi [5] for the Pentium 4.

We selected four benchmarks (bzip2, gap, gcc and parser) from the SPEC CPU2000 benchmark suite [26], since these benchmarks show more temperature differences than other benchmarks during the execution. Since running single benchmark of these four benchmarks does not increase the temperature so much, we would like to run two benchmarks at the same time. However, running two benchmarks on two threads sometimes defers reading the performance counters severely and incurs thermal throttling by the Pentium 4 processor, resulting in inefficient evaluation of scheduling techniques. To prevent the inefficiency, we schedule the tasks off-line instead of on-line. We ran two applications separately and obtained the trace of the activity factor of all functional blocks. After then, we utilize off-line task scheduling, by using activity factor of all functional blocks. When the proposed technique using performance counters is adopted in the real world, the access to the performance counter can be set to have a higher priority than the other tasks in order to allow periodic accesses to the performance counter.

By running applications, we can have the coefficients for the formula. For more accurate estimation, we only use the samples whose IIPC is more than 2.0. We set the confidence interval is 99% in order to cover as many cases as possible. The formula that we obtained from the simple regression analysis is $Y = 14.1*X + 58.4$, where the IIPC (X) corresponding to 95 Celsius (Y) is 2.59.

The DFS using thermal sensors lowers the frequency by 20% when the temperature is same as (or more than) 95 Celsius. It increases the frequency by 5% every 10 ms up to the 2.6 GHz when the temperature is lower than 95 Celsius. The DFS using performance counters lowers the frequency to (2.6 GHz * (2.59/previous IIPC)), when the IIPC is more than 2.59. When the IIPC is lower than 2.59, the frequency is 2.6 GHz.

5 Evaluations

We evaluate the proposed DFS scheduling technique in six cases: bzip2 + gap, bzip2 + gcc, bzip2 + parser, gap + gcc, gap + parser, and gcc + parser. According to [25], maximum temperatures are between 65~100 Celsius in commercial microprocessors, depending on the model. We set the threshold temperature to 95 Celsius. We also assume that the frequency can be freely set not to distort the experiment results by discrete frequency.

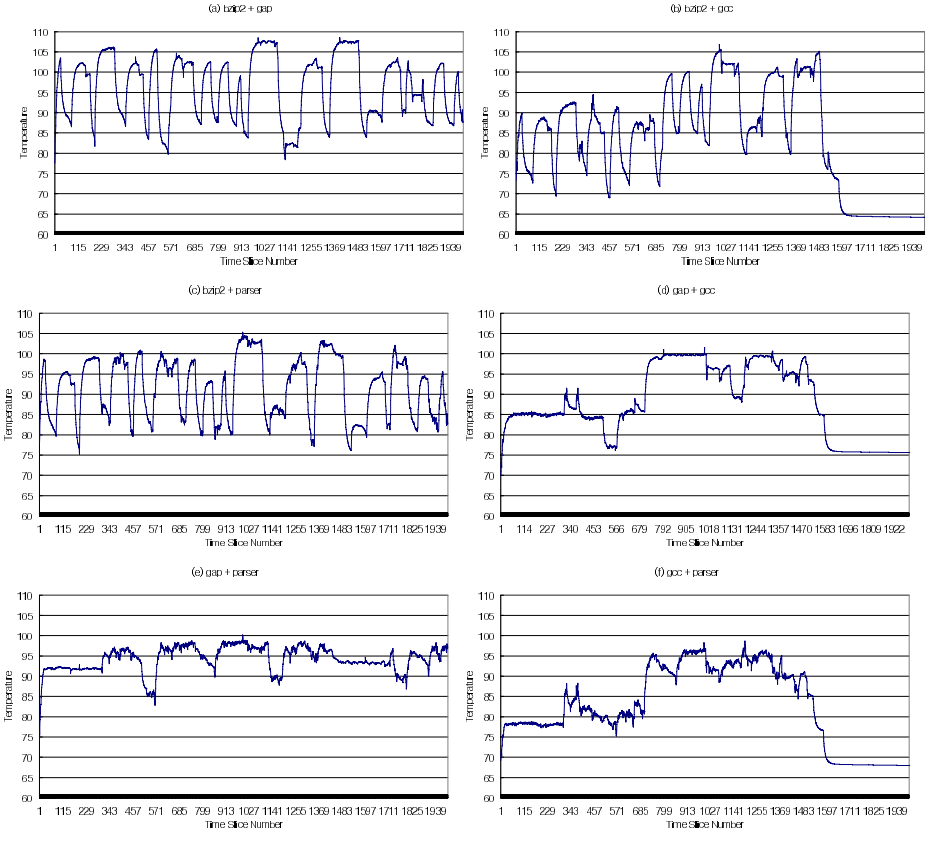


Fig. 1. Temperature changes (w/o DFS)

5.1 Scheduling Efficiency

Figure 1 shows the temperature changes when there is no consideration for temperature. In Figure 1, the temperature varies fast in (a), (b) and (c) due to the characteristic of *bzip2*, whereas the temperature does not vary so much and it is under 100 Celsius in (d), (e) and (f).

Figure 2 shows the temperature changes when DFS using CMOS thermal sensors is applied. As shown in the Figure 2, the temperature is varied significantly when the temperature is around 95 Celsius. The reason is that the frequency increases/decreases by a constant rate (20% for increase and 5% for decrease). If the frequency is decreased only by 10% or less, the temperature remains over 95 Celsius for longer time. When the frequency is increased more gradually, the performance loss will be severe. If the frequency is increased more than 5%, there are more temperature violations. Please note that there is no run-time information on how much the frequency should be changed. In fact, we tried to make use of the temperature history to find patterns of temperature variation in order to utilize the run-time information, which turned out not so helpful.

Figure 3 describes the temperature changes when the DFS is applied using performance counters. Different from Figure 2 where CMOS thermal sensors are used, Figure 3 does not show the spikes and dips of the temperature around 95 Celsius. In the proposed technique, the frequency is determined by referencing to the previous IIPC. When the previous IIPC is more than 2.59, the clock frequency is 2.6 GHz *

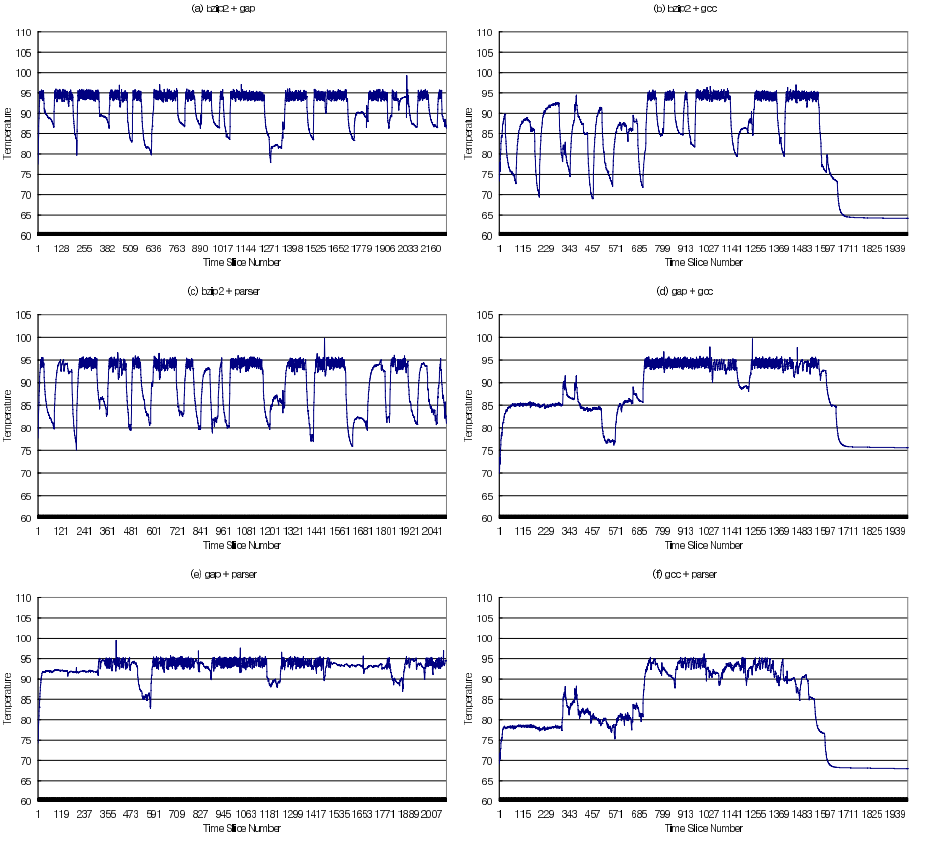


Fig. 2. Temperature changes (w/ DFS using thermal sensors)

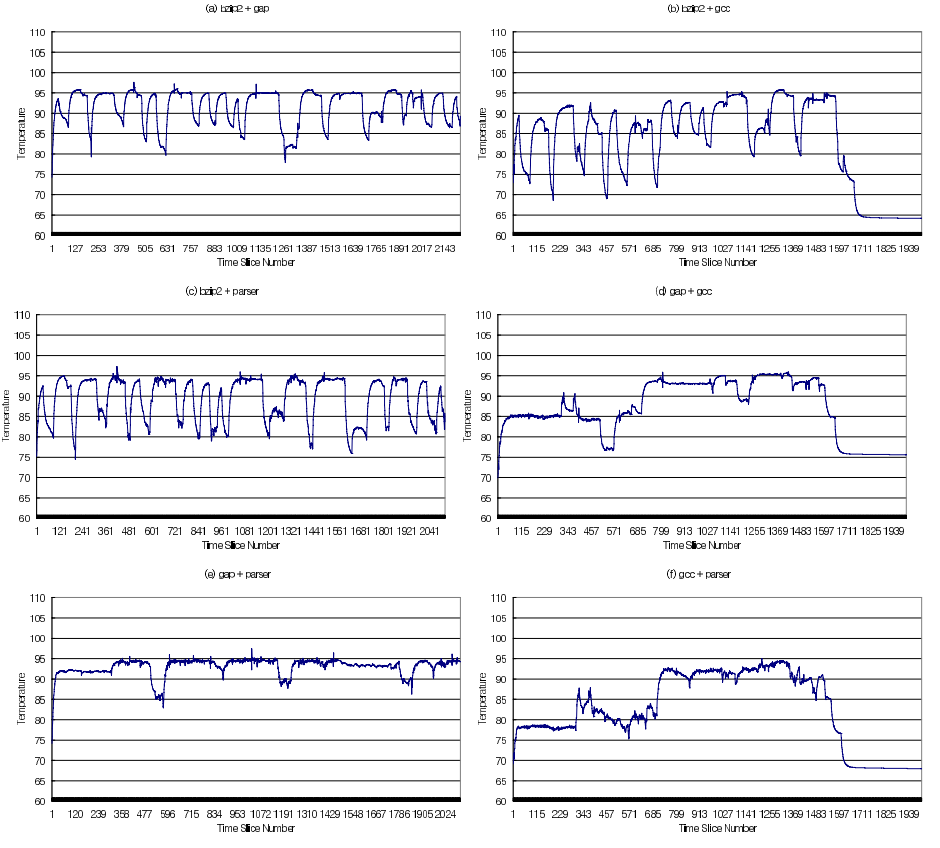


Fig. 3. Temperature changes (w/DFS using performance counters)

(2.59/(previous IIPC)). Otherwise, the frequency is 2.6 GHz (full speed). Thus, the fluctuation around 95 Celsius is less severe, compared to the DFS using thermal sensors.

As explained in the Section 3, using performance counters can make it possible to foresee the temperature tendency in advance. Accordingly, the proposed technique decreases the frequency early when the temperature goes up, which reduces the spikes around 95 Celsius.

5.2 More Details of Temperature Changes

Figure 4 presents the ratio of times when the actual temperature is over the threshold temperature. Both DFS techniques dramatically reduce the thermal violations. Sometimes the DFS using the performance counters performs better and sometimes does not. At least, we can say that the DFS using the performance counters is comparable to the DFS using the thermal sensors.

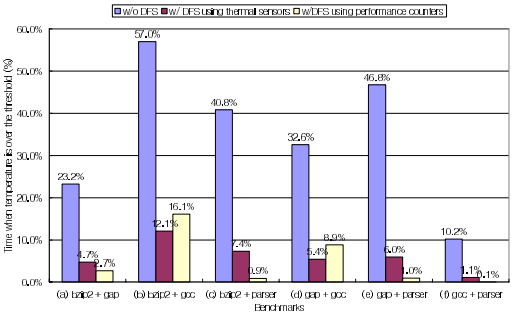


Fig. 4. Ratio of times when the actual temperature is over the threshold value (95 Celsius)

Figure 5 shows the average temperature difference between the actual temperature and the threshold value, when the actual temperature is over the threshold value. Though the temperature violation ratios in Figure 4 are not negligible, the average temperature excesses are significantly reduced. The average values of the temperature excesses in Figure 5 are 0.37 and 0.40 degree, on average, for the DFS using thermal sensors and the DFS using performance counters, respectively.

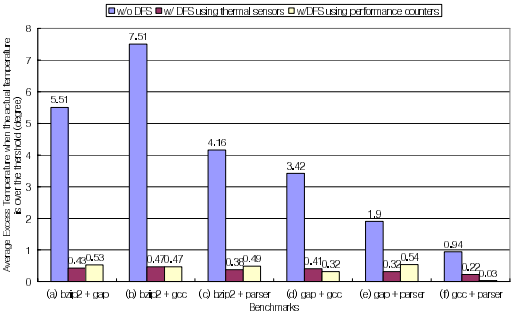


Fig. 5. Average temperature difference between the actual temperature and the threshold value (95 Celsius)

Figure 6 shows the maximum temperature when the actual temperature is over the threshold value. We can notice that the DFS using performance counters always outperforms the DFS using thermal sensors. The DFS using performance counters more accurately forecasts the temperature by referencing to the IIPC, which prevents the spikes. However, the DFS using thermal sensors can not predict future temperature. Thus, the temperature continues to go up even with the DFS, because the power

5.3 Performance

The tasks in this experiment are not periodic, in other words, which is not predictable. Thus, we should sacrifice the performance to sustain the temperature under the threshold value. If more aggressive DFS technique were adopted, the number of thermal violations would be decreased. As the number of thermal violations

decreases, the performance is naturally degraded. For example, suppose that one technique sets the threshold value to 90 Celsius and the other sets it to 100 Celsius. The former has less thermal violation and more performance degradation. For a fair comparison, we should check that the both techniques are similarly aggressive. If the proposed DFS using performance counters performed much worse than the DFS using thermal sensors, the experiment would not be fair.

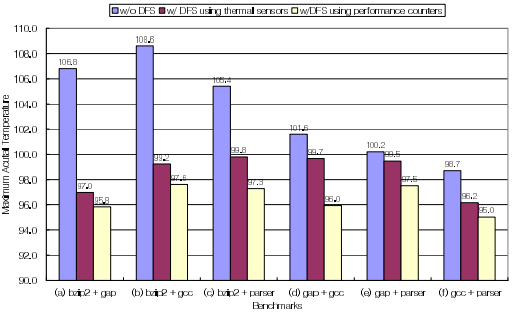


Fig. 6. Maximum temperature for each technique consumed in the past should be dissipated, resulting in higher maximum temperature

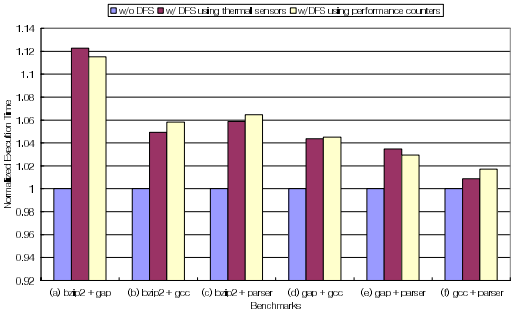


Fig. 7. Execution time normalized to the no DFS

Figure 7 shows the execution time normalized to the no DFS. The relative execution time, compared to the no DFS, only depends on the benchmarks' characteristics, themselves. The importance lies in the relative execution time between the DFS using thermal sensors and the DFS using performance counters. As shown in Figure 7, it is hard to say which technique is better in terms of performance, which implies two techniques are similarly aggressive, in the perspective of thermal control.

6 Conclusions and Future Works

Uneven activity from one functional block to another, results in localized hotspots that may move over time. Thus, accurate thermal monitoring therefore requires lots of

thermal sensors. This may be too costly, because precise CMOS thermal sensors are expensive in terms of area and power. As an alternative, we can use performance counters and regression analysis.

In this paper, we show that the DFS using performance counters is comparable to (sometimes better than) the DFS using thermal sensors. The DFS using performance counters only have to utilize the performance counters that are already embedded in most commercial microprocessors. Especially, after fabrication, when a microprocessor or an SOC (System On Chip) turns out to have localized hotspots that are not covered by CMOS thermal sensors, the proposed technique using performance counters can be a cost-effective solution. Though we used the temperature from the Hotspot [13][14] for regression analysis, the temperature from more accurate circuit-level thermal simulations can be used for regression analysis, which leads to more efficiency.

We only concentrated on the integer register file. However multiple functional blocks can be monitored and controlled using performance counters, since different clock frequencies might be assigned to different functional blocks. In this paper, we freely change the frequency but experiments with discrete frequencies would be interesting. We only examined the scheduling efficiency only with the DFS, since the DVS is not so reliable due to technology scaling [6] and it has more timing overhead [11]. The alternative to the DFS is clock gating to cool down the localized hotspots.

Acknowledgements

This works was supported by a Korea University Grant.

References

1. F. Bellosa, A. Weissel, M. Waitz, and S. Kellner. Event-Driven Energy Accounting for Dynamic Thermal Management. In Proceedings of COLP 2003, Sep. 2003.
2. D. Brooks and M. Martonosi. Dynamic Thermal Management for High-Performance Microprocessors. In Proceedings of HPCA'01, Jan. 2001.
3. S. W. Chung and K. Skadron. Using On-Chip Event Counters for High-Resolution, Real-Time Temperature Measurements. Proceedings of ITHERM'06, June 2006.
4. M. Huang, J. Renau, S.-M. Yoo, and J. Torrellas. A Framework for Dynamic Energy Efficiency and Temperature Management. In Proceedings of Micro'00, 2000
5. C. Isci and M. Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical data. In Proceedings of Microarchitecture (Micro'03), Dec. 2003.
6. V. Narayana and Y. Xie. Reliability Concerns in Embedded System Designs. IEEE Computer, vol. 39 no. 1, pp.118-120, Jan. 2006.
7. K.-J. Lee and K. Skadron. Using Performance Counters for Runtime Temperature Sensing in High-Performance Processors. In Proceedings of the Workshop on High-Performance, Power-Aware Computing (HP-PAC), April 2005.
8. K.-J. Lee and K. Skadron. Analytical Model for Sensor Placement on Microprocessors. In Proceedings of the IEEE International Conference on Computer Design (ICCD'05), Oct. 2005.
9. Merkel, F. Bellosa, and A. Weissel. Event-Driven Thermal Management in SMP Systems, In Proceedings of the Second Workshop on Temperature-Aware Computer Systems (TACS'05), June 2005.

10. M. D. Powell, M. Gomaa, and T. N. Vijaykumar. Heat-and-Run : Leveraging SMT and CMP to Manage Power Density Through the Operating System. In Proceedings of International Conference on Architectural Support for Programming Language and Operating System (ASPLOS'04), Oct. 2004.
11. E. Rotem, A. Naveh, M. Moffie, and A. Mendelson. Analysis of Thermal Monitor Features of the Intel Pentium M Processor. In Proceedings of the Second Workshop on Temperature-Aware Computer Systems (TACS'04), June 2004.
12. K. Skadron, T. Abdelzaher, and M. R. Stan. Control-Theoretic and Thermal-RC Modeling for Accurate and Localized Dynamic Thermal Management. In Proceedings of HPCA'02, Feb. 2002.
13. K. Skadron, M. Stan, W. Huang, S. Velusamy, K. Sankaranarayana, and D. Tarjan. Temperature-Aware Microarchitecture. In Proceedings of the 30th International Symposium on Computer Architecture (ISCA'03), June 2003.
14. K. Skadron, M. Stan, K. Sankaranarayana, W. Huang, S. Velusamy, and D. Tarjan. Temperature-Aware Microarchitecture: Modeling and Implementation. ACM Transaction on Architecture and Code Optimization. Vol. 1, No. 1, March 2004, pp. 94-125.
15. B. Sprunt. Pentium 4 Performance-Monitoring Features. IEEE Micro, 22(4), Jul/Aug 2002.
16. J. Srinivasan and S. V. Adve. Predictive Dynamic Thermal Management for Multimedia Applications. In Proceedings of International Conference on Supercomputing (ICS'03), June 2003.
17. Weissel and F. Bellosa, Dynamic Thermal Management for Distributed Systems. In Proceedings of the First Workshop on Temperature-Aware Computer Systems (TACS'04), June 2004
18. Apple Computer. Quad G5 2.5Ghz Processors. Available in http://homepage.mac.com/thunderaudio PhotoAlbum11.html.
19. J. Citaerlla. The Intel PIV's Thermal Diodes. Available in http://www.overclockers.com/artocles 517.
20. HP Corporation, Intel Corporation, Microsoft Corporation, Phoenix Tech. Ltd., and Toshiba Corporation, "Advanced Configuration and Power Interface Specification",. Available in http://www.acpi.info /DOWNLOADS/ACPIspec30.pdf, September 2004
21. Intel Corportation. IA-32 Intel Architecture Software Developers Manual. Vol. 3: System Programming Guide, 2004.
22. Intel Pentium 4 Northwood Die Photo. Available in http://www.chip-architect.com/news/003_04_20_Looking_at_Intels_Prescott_part2.html
23. Intel Pentium 4 Technical Documents. Available in http://www.intel.com/design/Petium4/documentation.html
24. Intel Corporation. Thermal Zone Information. Available in http://support.intel.com/support/motherboards/desktop/sb/CS-12552.htm
25. RCN Corporation. Processor Electrical Specifications, Available in http://users.erols.com/chare/ elec.htm
26. Standard Performance Evaluation Corp.. Available in http://www.specbench.org.
27. B. Sprunt. Brink and Abyss Pentium 4 Performance Counter Tools for Linux. Available in http://www.eg.bucknell.edu/bsprunt/emon /brink_abyss.

The Optimum Network on Chip Architectures for Video Object Plane Decoder Design

Vu-Duc Ngo, Huy-Nam Nguyen, and Hae-Wook Choi

System VLSI Lab, SITI Research Center, School of Engineering
Information and Communications University (ICU)
Yusong P.O. Box 77, Taejon 305-714, Korea
{duc75, huynam, hwchoi}@icu.ac.kr

Abstract. On Chip Network (OCN) has been proposed as a new methodology for addressing the design challenges of future massly integrated system in nanoscale. In this paper, three differently heterogenous Tree-based network topologies are proposed as the OCN architectures for Video Object Plane Decoder (VOPD). The topologies are designed in order to maximize the system throughput. This paper also evaluates the proposed topologies by comparing them to other conventional topologies such as 2-D Mesh and Fat-Tree with respects to throughput, power consumption and size. We use the power modelling tool, known as Orion model to calculate the static powers, areas, and dynamic energies of three topologies. The experiment results show that our Tree-based topologies offer similar throughputs as Fat-Tree does and much higher throughputs compared to 2-D Mesh while use less chip areas and power consumptions.

1 Introduction

According to [1], the Ultra Large Scale Integration (ULSI) will be the future of chip design. The idea of using Network on Chip (NoC) as the new design methodology to massly integrate the System on Chip (SoC) IPs such as processors, DSPs, as well as memory array, which was proposed in [2,3]. The packet switching core and the communication protocols are used to replace the complex system of wires, the main factor that leads to the propagation delay exceeding the system's clock period and non-scalable global wire.

The tile-based NoC with various applications and regular topologies 2D Mesh, Fat-Tree, and Torus were proposed in [4, 5, 14, 15, 16]. The authors in [6, 7, 8] had different approaches for the NoC design. They proposed the algorithms to automatically map IPs onto the target NoC architecture so as to optimize the power consumption. However, the performance of system can not be optimized (the trade-off in performance and power is still an open problem). Wang et al. [9] proposed a power-model simulator for interconnection network called Orion. This simulator can be used to estimate the dynamic power of one information bit due to the switched capacitances. It also can be used to estimate the router area and static power with different CMOS technologies.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 75–85, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Recently, the VOPD was designed and evaluated based on 2-D Mesh and Fat-Tree topologies [8, 10]. In [10], Nguyen et al. reported the system-level performance evaluations of two architectures by using NS-2 simulator. The optimal mappings that maximize the system throughput also were presented. According to that paper, with different routing and queuing protocols, the Fat-Tree topologies offers better performance. However, because of its irregularity and intensive interconnection wires, the Fat-Tree topology becomes impractical to be implemented on a chip. Moreover, design of NoC for particular application copes with several challenges [11], one of the critical issues is to choose the best suited architecture.

Therefore, in this paper we propose three different Tree-based architectures for the VOPD design and then compare them to 2-D Mesh and Fat-Tree architecture. Fig. 1 presents the VOPD's sub-modules and the pair-to-pair data transactions (in Mbps).

From/To	AC/DC	ARM	IDCT	Iqua	Invers	Run_Lenght	PAD	Str_MEM	VMEM	Vrecst	VarLen	UPSAM
AC/DC	0.0	0.0	0.0	362.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
ARM	0.0	0.0	16	0.0	0.0	0.0	16.0	0.0	0.0	0.0	0.0	0.0
IDCT	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	353.0
Iqua	0.0	0.0	357.0	0.0	0.0	0.0	0.0	49.0	0.0	0.0	0.0	0.0
Invers	362.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Run_Lenght	0.0	0.0	0.0	0.0	362.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
PAD	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	313.0	0.0	0.0	0.0
Str_MEM	27.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
VMEM	0.0	0.0	0.0	0.0	0.0	0.0	94.0	0.0	0.0	500.0	0.0	0.0
Vrecst	0.0	0.0	0.0	0.0	0.0	0.0	313.0	0.0	0.0	0.0	0.0	0.0
VarLen	0.0	0.0	0.0	0.0	0.0	70	0.0	0.0	0.0	0.0	0.0	0.0
UPSAM	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	300.0	0.0	0.0

Fig. 1. Data transaction between modules of VOPD

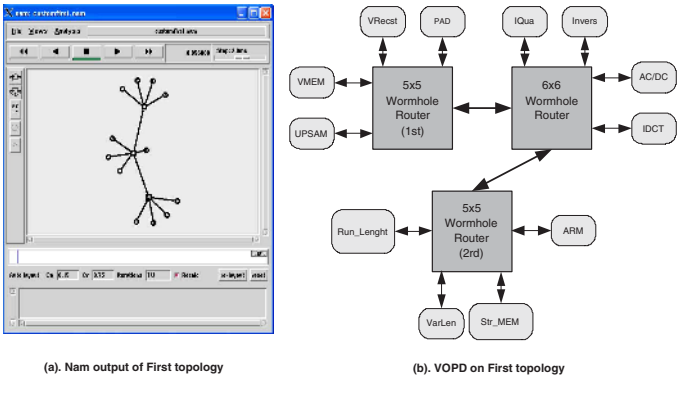


Fig. 2. VOPD on the first topology

We use NS-2 [12] to simulate the system throughputs of five architectures. By using these simulated throughputs and Orion model of utilized routers, the

system dynamic power consumptions of three architectures can be exactly calculated. The designs of three architectures also are evaluated and compared with each others and the two mentioned conventional architectures in terms of area and dissipated energy. Our results indicate that the proposed topologies are superior in terms of hardware consumption and energy consumption while offer similar system performances. The rest of this article is organized as follows: Proposed topologies construction is presented in Section 2. Router and power estimation are discussed in Section 3. Simulation model and analysis are introduced in Section 4. Finally, we conclude our contribution and mention about our future work in Section 5.

2 Proposed Topologies Construction

In this section, we present the construction of three Tree-based topologies and design the VOPD decoder by allocating its functional sub-modules onto the appropriate routers. Firstly, we briefly reintroduce two regular NoC architectures known as **SPIN** and **CLICHE**.

The **SPIN** architecture has been proposed by Guerrier and Greiner [4]. It uses Fat-Tree architecture to interconnect IP blocks. In this Fat-Tree, every node has four nodes as its children nodes and this rule is replicated four times at every level of the tree. Let assume N is the number of IPs in the architecture, the size of the network grows as $(NlogN)/8$ and the number of switches is $3N/4$. The switches, except for the root switches, are modelled by 4×4 wormhole router. The number of routers depends on the depth of the tree.

The **CLICHE** architecture has been proposed by Kumar et al. [5]. This architecture consists of an $m \times n$ 2-D mesh of switches that interconnects $m \times n$ IPs allocated along with the switches. Each switch connects with its for neighboring switches and one IP. Hence, a switch can be modelled by 5×5 wormhole router. The number of routers is also $m \times n$.

In [10], the authors pointed out that even with the optimal design the 2-D Mesh still offers lower performance in comparison with the Fat-Tree architecture. In that paper, the VOPD was designed with the 4×4 Mesh and the Fat-Tree of 16 IPs size. As shown in Fig.1, the number of IPs of the VOPD decoder is 12. Hence, there are several unused switches for both architectures. In other words, if we design this particular application on the generic architectures such as 2-D Mesh and Fat-Tree, we have to pay a big penalty on unused hardware. Due to the redundancy of unused switches in designing the VOPD of the two mentioned architectures, in this paper, we propose three Tree-based architectures in which the optimal allocation schemes of IPs are implemented. These topologies probably are the best suited architectures for the VOPD.

First topology: As shown in the Fig. 2, the 12 functional blocks (IPs) of the decoder are mounted onto three wormhole routers, two are 5×5 and one is 6×6. For this architecture as well as the remaining two architectures, we use similar Branch and Bound technique presented in [10, 14] to achieve the optimal mapping (it was applied for 4×4 Mesh architecture). This work is done by knowing

the required data transactions between IPs and the routing table. In this article, we also apply the shortest path routing algorithm for the simplicity and practice. This optimal mapping makes the network satisfy the condition of obtaining the highest network throughput. As the result shown in Fig. 2, the UPSAM, VMEM, VRecst, and PAD are interconnected with the first router. The second router interconnects IQua, Invers, and IDCT. These IPs, as depicted in the Fig. 1, are the functional blocks that have the highest data transactions from one to the others. This also means that we can reduce the packet's drop rate as well as the huge amount of data transaction between two routers. Straightforwardly, if we apply the power model as introduced in [9], then the power dissipation of the network (in both router and wire) can be significantly reduced. The remaining low data rate IPs, such as ARM, Run_lenght, Str_MEM and VarLen, are mounted onto the third router. This fact is going to discuss in the next section.

Second topology: Fig. 3 shows that 12 IPs of the VOPD are optimally mounted on four different wormhole routers. The highest data transaction IPs such as UPSAM, VMEM, VRecst, PAD, IDCT, IQua and Invers, are still mounted on two neighboring routers to obtain highest network throughput as well as smallest power dissipation.

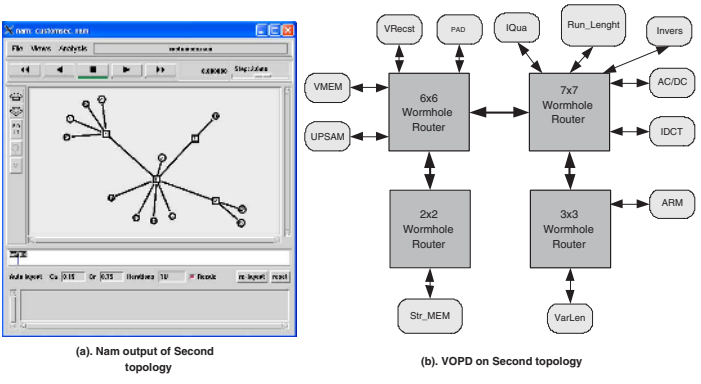


Fig. 3. VOPD on the second topology

Third topology: This topology looks most likely Tree topology. As shown in Fig. 4, there are totally five routers including one 6×6, one 5×5 and three 3×3 wormhole routers. In this topology, a similar technique as applied in above two topologies is used. Hence, the optimal mapping of IPs onto OCN architecture can be seen as Fig. 4b.

3 Router and Power Estimation

This section introduces the model of wormhole routers which were mentioned in previous sections and their power models to obtain the network power consumption and areas of the three proposed topologies.

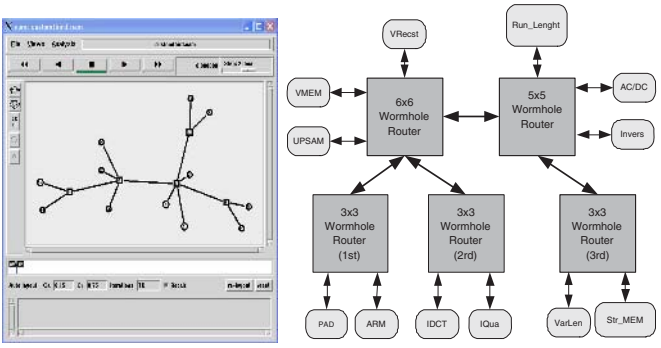


Fig. 4. VOPD on the third topology

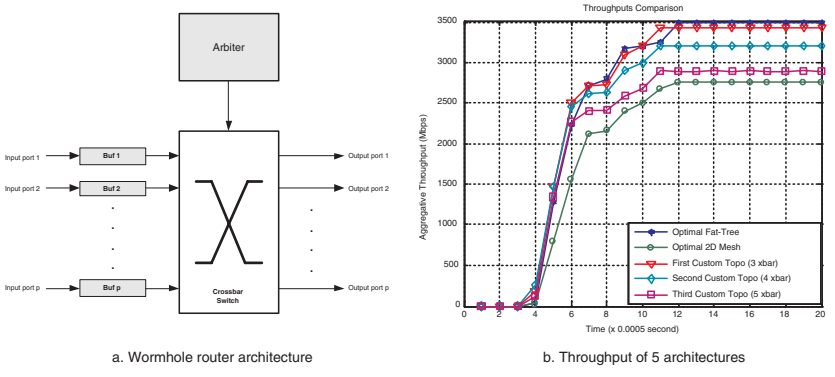


Fig. 5. Wormhole router model and Throughput comparison

As we can see in Fig. 5a, the wormhole router includes p input ports and p output ports. One or some of them can be used for the interconnections with the neighboring routers, the remaining ports are used for interconnection with IPs. The wormhole handles the dataflow on a flit level where flit is the smallest data unit. The data packet is divided to a number of flits where the head flit carrying the destination address and the tail flit are important for traversing whole packet through the router. For instant, when the source injects the head flit into input port i and this flit contains the destination address of output port j, firstly the buffer i writes the flit into the tail of it. Secondly, after the request of destination port is read and granted by the arbiter, the arbiter sends the signal to the crossbar switch to emit the data to the output port j by reading flits out of the buffer i. Hence, the energy dissipation of the flit on this router is given by

$$E_{flit} = E_{arb} + E_{xbar} + E_{bufrd} + E_{bufwrt}, \quad (1)$$

where E_{arb} is the arbitration energy, E_{xbar} is the crossbar switch energy, E_{bufrd} and E_{bufwrt} are the read and write to/from buffer energies, respectively. Since the dynamic power of CMOS circuit is calculated as

$$P = E \times f_{clk}, \tag{2}$$

where f_{clk} is the clock frequency and $E = \frac{\alpha}{2} C V_{dd}^2$ with α the switching activity, C the circuit capacitance, and V_{dd} the supply voltage. Therefore, if f_{clk} is defined as the router's operational frequency then the power dissipation of a flit on this router is formulated as

$$P_{flit} = \frac{1}{2} \alpha f_{clk} V_{dd}^2 \left(C_{xbar} + C_{arb} + C_{bufrd} + C_{bufwrt} \right). \tag{3}$$

To calculate this power dissipation we apply Orion model presented in [9]. For the homogenous architecture, the power dissipation on the router of whole network is generally given by

$$P = f_{clk} E = f_{clk} H_{avg} \left(E_{arb} + E_{xbar} + E_{bufrd} + E_{bufwrt} \right), \tag{4}$$

where H_{avg} stands for the average hop count of the network and it is very much related to the traffic pattern. For instant, the power dissipation on routers of $N \times N$ 2-D Mesh is presented as

$$P = f_{clk} \frac{2N}{3} \left(E_{arb} + E_{xbar} + E_{bufrd} + E_{bufwrt} \right). \tag{5}$$

In this paper, as we mentioned above, the five topologies are heterogenous in terms of configuration and operational frequency. The operational frequency of each router is decided by the highest data transaction IPs mounted onto it. Without loosing generality, the power dissipation of a particular flit is given by

$$P_{flit}^i = \sum_{R_i} f_{clk}^j \left(E_{arb}^j + E_{xbar}^j + E_{bufrd}^j + E_{bufwrt}^j \right) \times \delta_{ij} \tag{6}$$

where R_i is the known route that the flit i goes through, and

$$\delta_{ij} = \begin{cases} 1, & if j^{th} router \in R_i, \\ 0, & otherwise. \end{cases} \tag{7}$$

Finally, based on the simulated network throughput and the routing table, the power dissipation of the network with respect to routers is presented as

$$P_{Net} = \sum_{\forall i} P_{flit}^i. \tag{8}$$

In this paper, the wire energy dissipation, $E_{wire} = \frac{\alpha}{2} C_{wire} V_{dd}^2$, is calculated by using the wire model worked out [13]. More particular, we use $0.1 \mu m$ technology and supplied voltage of $1.2V$ for the repeated wire model.

To calculate the powers and energies for 2D Mesh, Fat-Tree and three proposed architectures, we use Orion power model for each individual router in each architecture. We apply $0.1\mu m$ technology, supply voltage of $1.2V$ and flit size of 128 bits for all routers. The operational frequency of each router depends on the IP which has the highest data transaction. Additionally, after calculate the power consumption we can also obtain the router's area of each architecture. Table. 1 presents the power consumption, dynamic bit energy and corresponding used area of three wormhole routers of the **first topology**.

Table 1. Power, bit energy and area of Routers in First Topology

	First 5×5 Router	6×6 Router	Second 5×5 Router
Area (μm^2)	993280	1.41e6	993280
Flit power (W)	0.153758	0.172508	130564
Bit energy (J)	2.40e-12	3.70e-12	2.81e-12

Table 2. Power, bit energy and area of Routers in Second Topology

	6×6 Router	7×7 Router	3×3 Router	2×2 Router
Area (μm^2)	1.41e6	1.91e6	374784	176128
Flit power (W)	0.204922	0.219783	0.042822	0.025032
Bit energy (J)	3.20e-12	4.70e-12	4.80e-12	7.20e-12

Table. 2 presents the power consumption, dynamic bit energy and corresponding used area of four wormhole routers of the **second topology**. The power consumption, dynamic bit energy and corresponding used area of five wormhole routers of the **third topology** are presented in Table. 3.

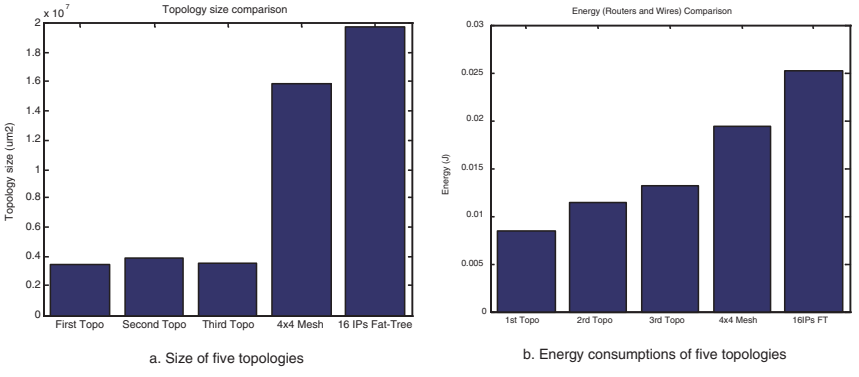
Table 3. Power, bit energy and area of Routers in Third Topology

	6×6 Router	5×5 Router	1^{st} 3×3 Router	2^{rd} 3×3 Router	3^{rd} 3×3 Router
Area (μm^2)	1.41e6	993280	374784	374784	374784
Flit power (W)	0.204922	0.130564	0.059266	0.062245	0.042822
Bit energy (J)	3.20e-12	2.805e-12	1.48e-12	1.36e-12	4.80e-12

4 Simulation and Analysis

In this article, the behavioral performance of the designs of VOPD on five architectures including three proposed Tree-based topologies, 2-D Mesh and Fat-Tree are simulated by NS-2. The simulations are done based on the data transaction between IPs depicted in Fig. 1. The transmission protocol is defined as UDP. The exponential traffic generator model is applied. The routing strategy is shortest path. To tolerate with the Orion model, we set the packet size of 64 bytes or four flits of 128 bits. The buffer scheme of DropTail is utilized. As the simulation shows in the Fig. 5b, our three newly Tree-based architectures offer not much less system throughputs in comparison with Fat-Tree and 800Mbps higher than 4×4 Mesh. Among three new architecture, the **first topology** is the best one in terms of system throughput.

Based on the area calculation in previous section, the total router sizes of five architectures are sketched in Fig. 6a.



a. Size of five topologies

b. Energy consumptions of five topologies

Fig. 6. Sizes and Energies of five Topologies

As shown in Fig. 6a, the Fat-Tree topology utilizes biggest chip area, size of $19.8mm^2$ with the $0.10\mu m$ CMOS technology. The second biggest one is 4×4 Mesh, size of $15.9mm^2$. While the biggest one among our three proposed topologies is the **second topology**, size of $3.87mm^2$. This figure also depicts that the router area of three proposed architectures are almost similar. From Fig. 5b and Fig. 6a, we can conclude that our three proposed topologies offer higher system throughputs than the optimal 2-D mesh while little smaller than the optimal Fat-Tree. The **first topology** offers the highest throughput because it has simplest connectivity or it can avoid more efficiently the drop packet at the intermediate nodes. The remaining two proposed topologies still offer higher throughputs compared to 2D Mesh while have significantly smaller sizes. This again indicates that two conventional architectures introduce big overhead of hardware consumption.

The simulated energy consumptions of five topologies are obtained basically on equations (6, 7, 8). Since we have the system throughput simulation results

of five topologies. We also know the in detail about the route of every individual flit. This means that we can point out which routers and wires the given flit goes through. Using the bit energy calculated in Table. 1, Table. 2 and Table.3, finally we can have the simulated energy consumptions of our proposed topologies as depicted in Fig. 6b. As shown in Fig. 6b, it is interesting that the **first topology** consumes smallest energy, of $0.0084J$, compared to two other proposed ones. Due to the complicated flows of flits and big number of switches of Fat-Tree and 2-D Mesh, it is obvious that if we use similar parameters for simulation, the energy dissipations on these two architectures are much bigger than our three proposed architectures. Additionally, the throughputs of three proposed architectures depicted in Fig. 5b indicate that the dropped flits at the intermediate routers of the second and the third topologies consume quite big amount of energy. Finally, we can indicate that the **first topology** consumes less power while offers higher system throughput compared to the remaining two proposed topologies. The portions of the energy dissipations insulted from wires of five topologies are depicted respectively in Fig. 7.The **first topology** causes smallest consumption because of its simplest connectivity. Likewise, the Fat-Tree induces highest consumption since it has most complicated connectivity.

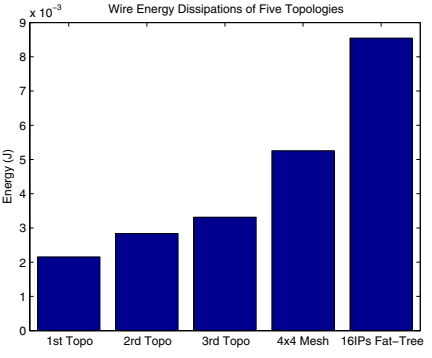


Fig. 7. Wire Energies of five Topologies

To show the overperformance of the optimal mappings of our proposed topologies, we also carry out the experiments of three cases of random mappings. The random mappings of our three topologies are set up by randomly interchanging the high data rate IPs with low data rate IPs in different routers. The throughputs of these three cases are depicted and compared with three cases of optimal mappings in Fig. 8. The corresponding consumed energies of random and optimal mappings of three architectures are shown in Fig. 9. It is interesting that the random mappings not only offer less throughputs but consume more power. The main reason for this contradiction is that the random mapping cases create more complicated routes and dropped flits. The more complicated route is, the

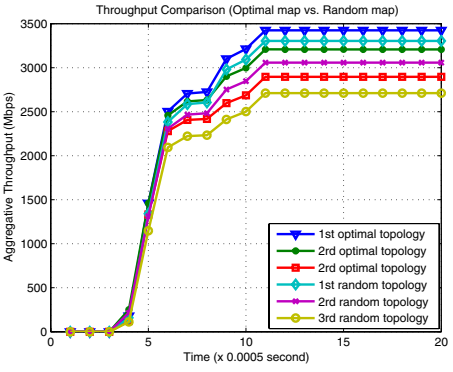


Fig. 8. Throughput Comparison: Optimal map vs. Random map

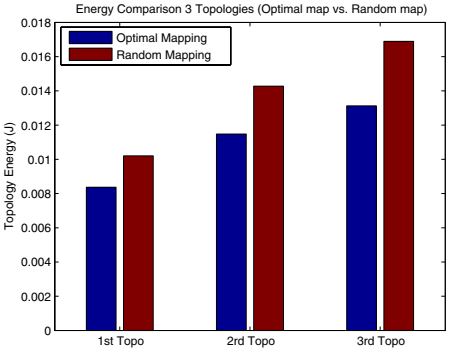


Fig. 9. Energy Comparison: Optimal map vs. Random map

more energy is consumed. Moreover, the total energy dissipation is calculated by accumulating flits's energy until they reaches the destination or drops out of the network.

5 Conclusion

In this paper, we designed the VOPD with three differently Tree-based architectures. These architectures were designed to not only obtain the highest system-level performance but also reduce the hardware consumption and the intensive interconnections. We also evaluated these architectures in terms of throughput, dynamic and static power consumptions. Then, we compared our proposed architectures to 2-D Mesh and Fat-Tree architectures. The results showed that the newly proposed architectures used less hardware and consumed less power while offered almost similar throughputs as the Fat-Tree did. The results also showed that our architectures overperformed the 2-D Mesh in both aspects of hardware complexity, power consumption and system performance.

References

1. ITRS. International technology roadmap for semiconductors - (2005) edition, http://public.itrs.net/
2. Benini, L. et al.: Networks On Chips: A new SoC paradigm. IEEE computer, Jan, (2002)
3. Bertozzi, D. et al.: Network on Chip Design for Gigascale Systems on Chips. In Zurawski, R. (Ed). Industrial Technology Handbook, CRC Press (2004) pp. 49-80.
4. Guerrier, P. et al.: A generic architecture for on-chip packet-switched interconnectio. In Proc of Design Automation and Test in Europe Conf, Aug (2000) 250-256
5. Kumar, S. et al.: A Network on Chip Architecture and Design Methodology. In Proc of . Int'l Symp. VLSI (2002) 117-124
6. Hu, J. et al.: Exploiting the Routing Flexibility for Energy Performance Aware Mapping of Regular NoC Architectures. In Proc of Design Automation and Test in Europe Conf, March (2003)
7. Hu, J. et al.: Energy-Aware Communication and Task Scheduling for Network-on-Chip Architectures under Real-Time Constraints. In Proc of Design Automation and Test in Europe Conf, Feb. (2004)
8. Murali, S. et al.: Bandwidth-Constrained Mapping of Cores onto NoC Architectures. In Proc of International Conference on Design Automation and Test in Europe, (2004) 896-901
9. Hwang, H. S. et al.: Orion: A Power Performance Simulator for Interconnection Networks. IEEE Micro, Nov (2002)
10. Huy-Nam Nguyen, Vu-Duc Ngo, Hae-Wook Choi.: Realization of Video Object Plane Decoder on On-Chip-Network Architecture. In ICESS05, Lecture Notes in Computer Science, Vol. 3820. (Dec 2005) 256-264
11. Dally, W. J. et al.: Route Packets, Not Wires: On Chip Interconnection Networks. DAC. (2001) 684-689
12. Ns2: www.isi.edu/nsnam/ns/.
13. Ho, R. et al.: The future of wires. Proceedings of the IEEE Vol. 89, Issue 4, April (2001) 490 - 504
14. Dally, W. J. et al.: Route Packets, Not Wires: On-Chip Interconnection Networks. In Proc of the 38th DAC, June (2001)
15. Vellanki, P. et al.: Quality-of-Service and Error Control Techniques for Network-on-Chip Architectures. In Proceedings of the Great Lakes Symposium on VLSI (2004)
16. Eisley, N. et al.: High-level power analysis for on-chip networks. In Proc of the 7th International Conference on Compilers, Architectures and Synthesis for Embedded Systems, September (2004)
17. Nurmi, J.: Network-on-Chip: A New Paradigm for System-on-Chip Design. In Proc of International Symposium on System-on-Chip, Nov. (2005)

A Hardware NIC Scheduler to Guarantee QoS on High Performance Servers*

J.M. Claver, M. Canseco, P. Agustí, and G. León

Department of Computer Science and Engineering,
University Jaume I, 12071-Castellón, Spain
{claver, canseco, agusti, leon}@icc.uji.es

Abstract. In this paper we present the architecture and implementation of a hardware NIC scheduler to guarantee QoS on servers for high speed LAN/SAN. Our proposal employs a programmable logic device based on an FPGA in order to store and update connection states, and to decide what data stream is to be sent next. The network architecture is connection-oriented and reliable, based on credit flow control. The architecture scales from 4 to 32 streams using a Xilinx Virtex 2000E. It supports links with speeds in the order of Gbps while, maintaining the delay and jitter constrains for the QoS streams.

1 Introduction

Nowadays, QoS requirements on high performance system/local area networks (SAN/LAN) make necessary the use of complex routers to offer QoS hardware support between all their components. The MultiMedia Router (MMR) architecture [1] is an example of this sort of routers, designed as an interconnection compact component to be used in cluster and LAN/SAN environments. However, this router requires a high number of virtual channels (VC) per port (128) and a complex scheduler associated with every link, which maintains and updates connection states, and selects packets to be sent.

A way to reduce router complexity and to optimize its design, while meeting QoS constraints, consists in improving traffic scheduling in servers [10]. The main idea is to reduce the number of buffers and VCs, taking advantage of QoS traffic sorting carried out by servers. Thus, both scheduling of routers and servers could work to wire speeds. To do that, it is necessary to use a hardware implementation to accelerate the scheduling process. However, the complexity of stream selection and priority updating poses an important implementation problem for scheduling a large number of streams using Gigabit links.

In order to achieve these goals we propose a NIC (Network Interface Controller) scheduler architecture for servers with QoS support. Previous efforts [11,13,9] have proposed priority queuing architectures for switches and NICs. Our architecture is inspired on link schedulers designed for the MMR router. A simplified form of this

* This work was partially supported by the Spanish CICYT grant TIC2003-08154-C06 and the European FEDER program.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 86 – 97, 2006.
© Springer-Verlag Berlin Heidelberg 2006

router has been completely implemented on an FPGA, denoted as Simple MMR (SMMR) [4]. The SMMR has been conceived as a prototyping platform, adapting MMR design ideas to a more realistic chip area and clock rate constraints. Furthermore, SMMR allows design alternatives for the evaluation and optimization of the initial MMR architecture [6]. A first prototype based on this architecture is proposed for a hardware traffic generator to test the SMMR.

Our NIC scheduler is inspired on a previous work [9], which formulates a hardware solution for DWCS scheduling [14], using an *in chip* network processor and an FPGA on a board. On the other hand, our NIC architecture is a structured design which allows posterior improvements and optimizations. Also, the network architecture used in our case study allows implementing more components of the NIC scheduler in an FPGA. Only few tasks are coordinated by the server host processor in our implementation. These tasks will be performed by an FPGA embedded network processor in a future version.

The remainder of this paper is structured as follows. In the following section the network architecture is presented. In section 3 we review the architecture of the QoS hardware scheduled server. The hardware architecture and implementation of the NIC scheduler is described in section 4. In section 5 we report details on the experimental prototype and the performance evaluation. Finally, section 6 summarizes the results of this work and overview future lines of research.

2 Network Architecture

Our server NIC scheduler is designed for high performance SAN/LAN and server clusters. These environments produce heterogeneous traffic streams, best effort (BE) and QoS (constant bit rate, CBR, and variable bit rate, VBR), which have different bandwidth and latency requirements. We consider a network architecture in which data packets are partitioned into *flits* (flow control units) of 1024 bits. Every flit is composed by 64 phits, where a *phit* (16 bits) is the minimum information unit transferred through a network link. The transmission bitwidth used internally in the NIC scheduler will be 32 bits, which is the same bitwidth that the external memory words.

The establishment of QoS connections is carried out using an adaptive seeking algorithm of minimum path based on backtracking and denoted as Exhaustive Profitable Back tracking (EBP) [8]. For this purpose, two control packets, connection and disconnection, are used. Flow control used in this sort of traffic is PCS (Pipelined Circuit Switching). Best effort traffic uses a connectionless-oriented scheme, and a Virtual Cut Through (VCT) switching control. A Credit-based flow control, as in Infiniband [12], is used to avoid channel saturation and data loss. This technique consumes link bandwidth, which can be reduced by increasing flit size, and requires much smaller buffers.

Link communications are serial and data are synchronized in blocks of 1040 bits ((64+1) phits or 1flit + 1phit) that we denote as “flit frame”. Thus, a flit frame is divided in two parts, one corresponding to data, network control or synchronization flits, and another in which a credit may be sent.

3 Scheduler Architecture

A NIC scheduler for a server with QoS support must maximize link throughput, at the same time that guarantees real time requirements and avoids starvation of BE traffic. In the design of our NIC scheduler, we take into account the following strategies in order to achieve these goals: an admission control and bandwidth distribution to maximize the link throughput, an adaptive priority system to obtain QoS guarantees, and an inhibition mechanism on QoS streams, which avoids starvation of best effort traffic and controls link bandwidth usage.

In order to parameterize bandwidth required by all sorts of traffic (VBR, CBR and BE), time space is divided in multiples of a flit frame that we denote as "flit round" or just "round". The number K of flit frames per round determines the minimum bandwidth assigned to a traffic flow ($1/K$). In our experiments we set K=2048.

Admission and bandwidth assignment policies vary in function of traffic characteristics. For CBR traffic, only the bandwidth used is needed, which cab be expressed as the maximum number of flits that can be sent in a round. As VBR traffic bandwidth can vary from one round to the next, a statistic model is used to obtain medium and peak bandwidths, and to determine an adequate bandwidth reservation. Finally, remaining bandwidth is assigned to best effort traffic.

In this first scheduler model, VBR connexions are considered as CBR. Thus, to avoid packet loss and overtake delay requirements that will appear with statistics models, its peak bandwidth (PBR) is used to reserve bandwidth in scheduler. An adaptive priority system that guarantees QoS requirements is based either in delay or jitter. In this case, a priority algorithm based on bandwidth and delay has been selected, the SIABP (Simple-IABP) algorithm [2]. The algorithm has good behavior under high traffic loads and it is the core of the link scheduling algorithm in the MMR router [3]. The algorithm also increases the priority of packets proportionally to its waiting time in the buffer input queue. This design needs a regulating mechanism to control the correct use of bandwidth reserved for every stream, avoiding starvation of BE traffic. Therefore, for CBR traffic, the deadline to send a flit (T_Delay) is determined as a function of its transmitting pace (BW_{link}/BW_i, where BW_{link} is the link bandwidth and BW_i is the bandwidth reserved for the VC_i) in a round and its accumulated delay.

Taking into account the previous parameters, the order in which streams (S_i) are served follows Algorithm 1. Among all QoS streams ready to be sent – those with remaining bandwidth in the current round ($BW_R > 0$), buffer space in the receiver VC (credits), and satisfying its bandwidth constrains ($T\_Delay \leq 0$) – the highest priority stream is select to be sent. When there is more than one stream with the highest priority, any one of these streams is selected. The implementation of this algorithm is widely detailed in section 4.

A QoS server needs a high speed I/O interface to connect the storage system with the NIC scheduler board, providing the required bandwidth for every stream. New standards as the PCX commuted interface could provide the necessary bandwidth for current link bandwidths. Figure 1 shows the system architecture proposed for our QoS hardware scheduled server.

The system architecture is organized in two levels. The first level is carried out by server host processor, which provides first stage of admission control, initiate connection setup, stream identification and scheduling of stream buffers.

Algorithm 1. Scheduler stream selection

```
for all streams Si∈{CBR}, i=1,2,…,n /*In parallel
/*Builds the priority vector V
   if {Si.T_Delay≤0 & Si.Credits>0 & Si.BWR>0}
      Vi= Si.priority; /*Si participates in scheduling
   else
      Vi= 0; /*Si does not must send data
   Endif
end for
/*Selects the highest priority stream in log2n steps
{VH}= {highest(Vk)}, ∀Vk∈V; /*Vk with the high priority
if {card({VH}) == 1} Out= H;/*There is only one VH
else Out= Any({H});/*Selects any one of the highest VH
endif
```

The second level of this architecture is completely implemented on an FPGA, coordinating admission control with the host processor, managing stream scheduling, formatting and transmitting flits from stream buffers. So, the FPGA not only performs scheduling tasks, but also provides real time functions that were previously assigned to a network processor.

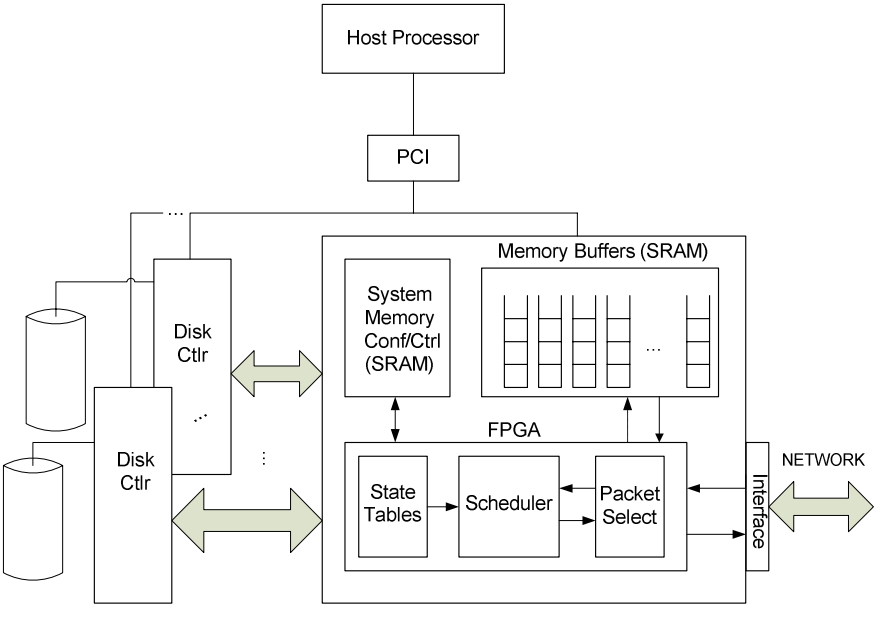


Fig. 1. System architecture of a QoS hardware scheduled server

This distribution of tasks is due to timing strong requirements. Management of scheduling needs acceleration due to the temporal bounds and the fact that the communication with the network interface demands strict synchronization. Alternatively, this could be implemented using an ASIC. However, to keep pace with the constantly changing algorithms, standards and protocols, it is preferable to use reconfigurable logic.

4 Hardware Architecture and Implementation

The hardware architecture and implementation of our NIC scheduler combine the use of an FPGA board, a network interface, and a system of disks. The components implemented in the FPGA perform several tasks: communication with the server host processor, stream scheduling, and data input/output (see Figure 2).

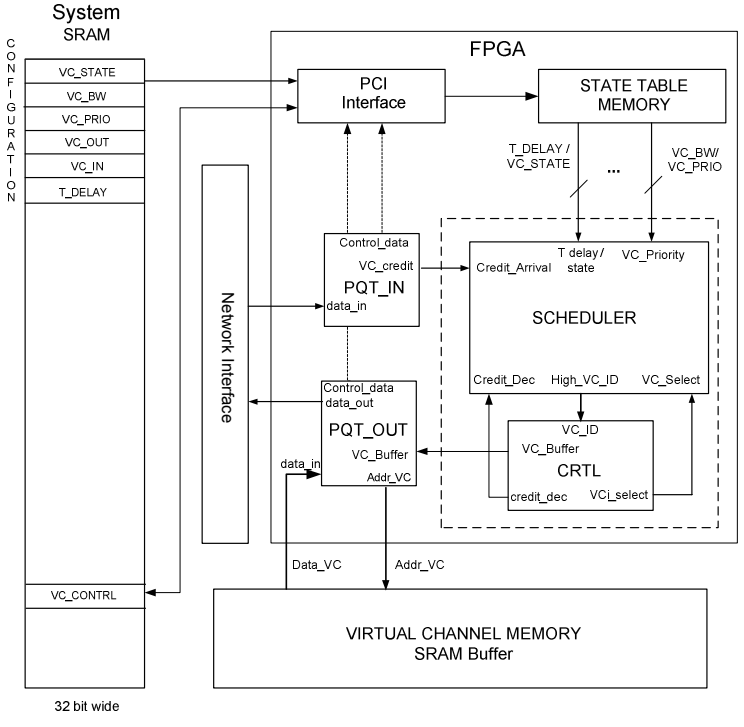


Fig. 2. Internal hardware and interfaces of an FPGA Server NIC scheduling prototype

Connection setup is carried out by the host processor, loading parameters such as stream type (CBR, VBR), reserved bandwidth, transmitting pace (T_Delay), and VCs involved on a stream. As data packets are transmitted, a bidirectional communication host-FPGA controls VC buffers, interchanging data and control information. A flit arrival through the network interface is immediately communicated to the host. The

host processor also indicates to the FPGA the end of a communication and updates its state. This is followed by the FPGA sending a disconnection flit. As it is described in section 2, communications are synchronized using a "flit frame", composed by 64+1 phits.

The input module (PQT_IN) controls the input link, identifying received packets: control flits, synchronization flits, and flow control credits. When a control flit arrives, the PQT_IN resends this to the host. When a synchronization flit is received, PQT_IN does nothing. In the last cycle of a flit frame a credit can be received from the router connected to the server. Then, the PQT_IN module informs of the credit arrival to the host through the PCI interface and to the SCEDULER module as well, increasing the corresponding credit counter of this stream.

The output module (PQT_OUT) builds the head of a flit and sends it, followed by data from the corresponding VC buffer. PQT_OUT obtains data directly addressing VCs stored on the SRAM memory of the FPGA board. For each flit that is sent, PQT_OUT informs the host processor to manage its associated VC buffer.

The scheduling algorithm is divided in two steps. First, all the sub-schedulers work in parallel to generate a global priority vector to be sent, where every sub-scheduler contributes with its VC identification and current priority. Second, the high priority VC is selected from the priority vector by a MAX bitonic network module (see Figure 3).

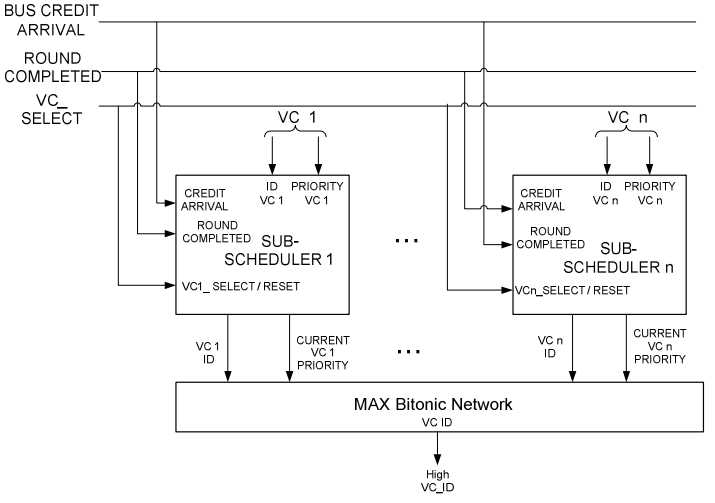


Fig. 3. Scheduler architecture composed of a sub-scheduler for every QoS data stream and of a MAX bitonic network

This proposal employs an implementation of a local scheduler (sub-scheduler) for every VC, which is composed of two modules, working synchronously in parallel (Figure 4). The SIABP priority module [2, 6] stores the priority of a VC, which is directly related to its bandwidth reservation as the number of flits that can be sent per round. After a scheduling process, all the priorities are updated. If a VC is selected to send its flit into the next flit frame, its priority is reset. Otherwise, its priority is increased as a function of its waiting time (for further details, see [1]).

The candidate selection is performed by a bitonic network (MAX) that selects the high priority VC from the priority vector generated by the sub-schedulers. The MAX module takes $\log_2 n$ cycles and has a building cost in LUTs area of O(n), where n is the maximum number of VCs per output link. It is possible to reduce the area of this module to either O($n/2$) or O($\log_2 n$), building a recirculating shuffle-exchange network as in [9]. Nevertheless, the resulting critical data path and complexity is then increased due to the presence of new multiplexers. This complexity is critical for the architecture of an FPGA, the building area is not significantly reduced, as the resulting delay clock is increased.

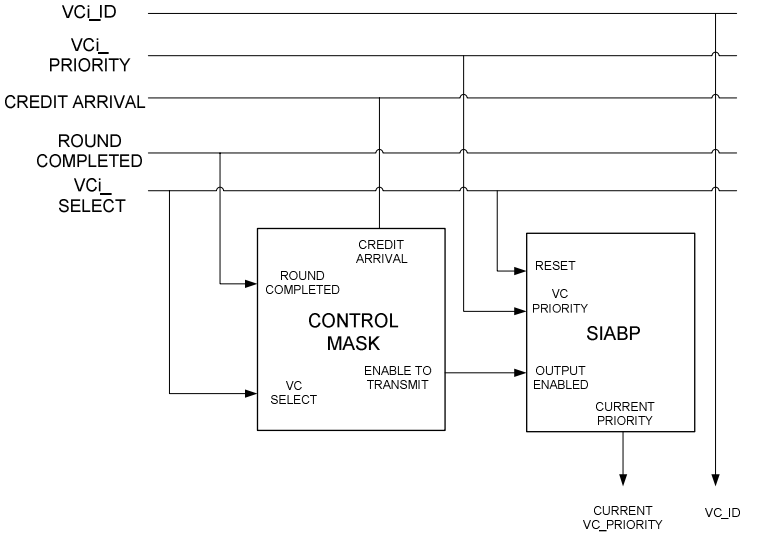


Fig. 4. Sub-scheduler architecture composed of a control module (CONTROL MASK) and of a SIABP priority module

The CONTROL MASK module (see Figure 5) controls the remaining bandwidth of a stream, the bank of credits, and the transmitting pace using the parameter T_DELAY. Using this information, this module decides if a VC must appear in the priority vector. If a VC has its CBR RATE counter set to a value less than 1, its BW REMAINDER counter greater than 0, and its CREDIT BANK counter greater than 0, the signal ENABLE TO TRANSMIT is set. At the same time the module adds its current priority to the priority vector. The priority added is zero only when one of the previous conditions is not satisfied.

The control module (CTRL) is the last step in the scheduling process. It synchronizes the scheduler and PQT_OUT modules, and actives signals to update internal registers and scheduler counters. Thus, the total router cycles required for scheduling are $10 + \log_2 n$, where the 5 cycles are devoted to local stream scheduling, $\log_2 n$ cycles are spent on stream selection and 5 cycles more are consumed updating stream priorities.

5 Evaluation

This section presents the performance evaluation of our NIC scheduler architecture and its hardware implementation using a Celoxica RC1000 board [7]. This board has an FPGA Xilinx Virtex 2000E (38,400 LUTs and 640 Kb BRAM), 8 MB SRAM memory distributed in four independent banks, and two PCI Mezzanine (PCM) I/O cards.

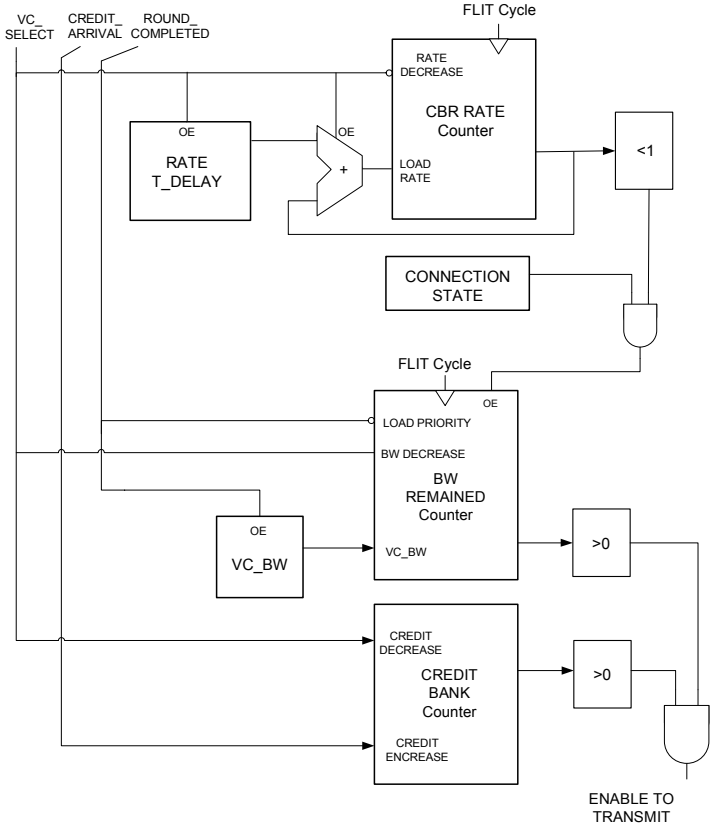


Fig. 5. Control module architecture (CONTROL MASK)

The components implemented into the FPGA carry out several tasks: communications with the host processor, scheduling of streams and data input/output. Communications between the FPGA and the host processor use DMA transferences between the host/PCI peer and the 32bit/33 MHz PLX controller of the RC1000 board.

In order to simplify the design process, the FPGA Virtex has been programmed using HandelC [5] and the Celoxica DK4 environment, and the ISE 6.1 Xilinx backend tools from Xilinx. This eases prototyping our design, and the updating

process. HandelC is a behavioral C based hardware description system developed by Celoxica that allows co-simulation. Parallelism of process and synchronization are taken from the CSP model, in particular, from the Occam language. HandelC uses standard data types with user defined bitwidths. Thus, HandelC provides an efficient use of hardware resources.

5.1 Area and Delay Results

For this evaluation we have scaled our design from 4 to 32 virtual channels. Thus, we can evaluate its behaviour and performances with a different number of streams. Figure 6 reports the area size and clock rate for 4, 8, 16, and 32 data streams. SRAM memory used for interface and VC buffers are not included in this table. As the number of streams handled by the NIC scheduler increase from 4 to 32, the number of LUTs utilized by the implementation grows linearly.

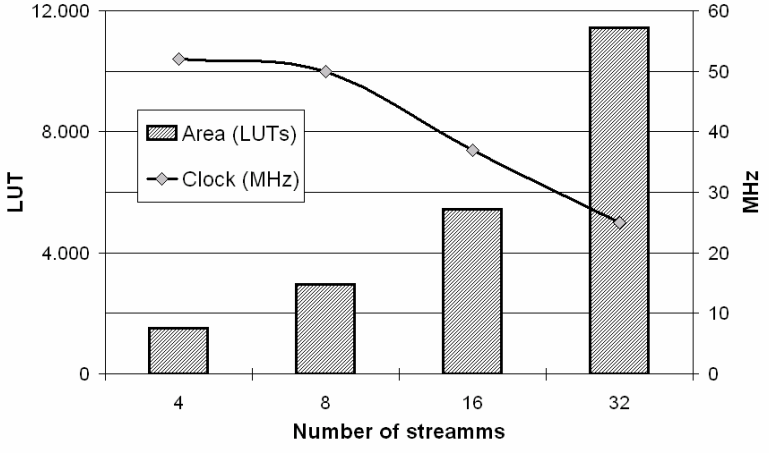


Fig. 6. Area and clock rate characteristics for different QoS server configurations

The maximum clock rate for a 4 streams configuration is 52 MHz, which decrease down to 25 MHz for the configuration with 32 streams. A detailed time analysis shows that maximum clock rate is limited by the sub-scheduler module and not by the selection module MAX. However, the maximum clock rate decreases logarithmically, showing that our NIC scheduler implementation can meet packet-time requirements on Gigabit links.

5.2 Traffic Analysis

In order to highlight performance results of our QoS NIC scheduler, we have compared it with another scheduler without QoS characteristics, based on a Round-Robin scheduling algorithm, and implemented in an FPGA too. We perform two experiments to illustrate bandwidth distribution among data streams. In experiment 1, four streams with equal bandwidth division ratios (1:1:1:1) were scheduled. In the

results the X-axis reports scheduled output time expressed in flit cycles (each flit cycle is equivalent to 65 router cycles), and the Y-axis reports the bandwidth as the number of flits scheduled per unit of time. We only report the first 100 flit cycles of simulation (65,000 router cycles). Figures 7 and 8 show the same bandwidth division for the two schedulers. The bandwidth of each stream settles after 70 time units and maintains the same value until the execution is terminated. So, bandwidth for every stream is guaranteed and output link throughput is 100%.

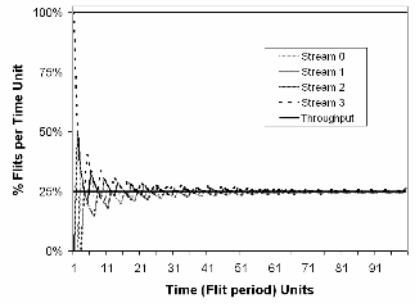


Fig. 7. Bandwidth distribution results for four streams with equal ratios (1:1:1:1) using the QoS scheduler

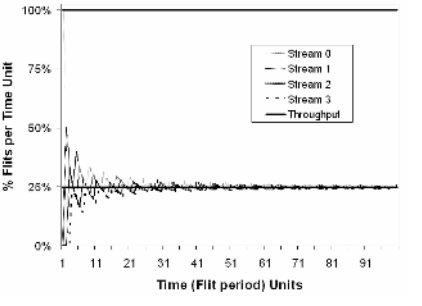


Fig. 8. Bandwidth distribution results for four streams with equal ratios (1:1:1:1) using a Round-Robin scheduler without QoS

A second experiment with bandwidth division ratios (4:2:1:1) was also recorded. Figures 9 and 10 report a different bandwidth distribution for the two schedulers. When the QoS scheduler is used, the bandwidth of each stream settles after 70 time units and maintains the same value until the execution is finished. So, bandwidth for every stream is guaranteed and output link throughput is 100%.

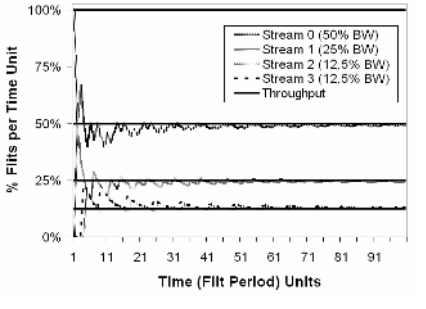


Fig. 9. Bandwidth distribution results for four streams with ratios (4:2:1:1) using the QoS scheduler

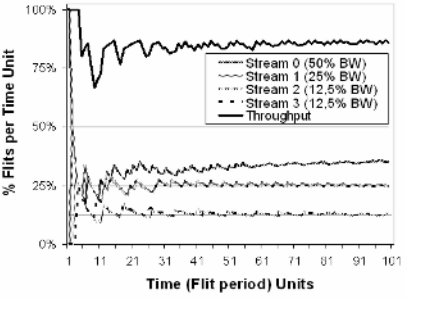


Fig. 10. Bandwidth distribution results for four streams with ratios (4:2:1:1) using a Round-Robin scheduler without QoS

However, as is reported in Figure 10, bandwidth for every stream is not guaranteed when the Round-Robin scheduler is used, and output link maximum throughput is

below 86%. Although bandwidth for streams with low requirements is guaranteed, bandwidth for the stream with high requirements is not.

6 Conclusions

In this paper we have presented the architecture and hardware implementation of a hardware NIC scheduler with QoS guarantees for servers on high speed LAN/SAN and high performance clusters. The proposed architecture is scalable because the occupation area grows linearly as the number of data streams is increased, and the NIC scheduler clock rate decreases logarithmically and tends to be stabilized around the 20 MHz. The QoS server behaviour guarantees bandwidth and delay requirements, attaining network link throughput close to 100%. NIC scheduler has been implemented in an FPGA and using a high level specification language like Handel-C.

Currently, we work in the design of a QoS server that handles combined CBR and VBR traffic with better efficacy and takes higher profit of maximum link throughput. Finally, we also work on increasing the clock rate of the NIC scheduler to adapt it to multi-gigabit data links.

References

1. Caminero, M.B.: Diseño de un Encaminador Orientado a Tráfico Multimedia en Entornos LAN. Phd. Thesis (in Spanish), Albacete, (Jun 2002)
2. Caminero, M.B., Carrión, C., Quiles, F.J., Duato, J., Yalamanchili, S.: A Cost-Effective Hardware Link Scheduling Algorithm for the Multimedia Router (MMR). Lecture Notes in Computer Science, Networking (ICN'01) (2001)
3. Caminero, M.B., Carrión, C., Quiles, F.J., Duato, J., Yalamanchili S.: Investigating switch scheduling algorithms to support QoS in the Multimedia Router (MMR). Proceedings, Workshop on Communication Architecture for Clusters (CAC'02), IEEE Computer Society (2002)
4. Canseco, M., Claver, J.M., León, G., Vilata, I.: Primeras Experiencias en la implementación de un encaminador QoS sobre una FPGA. IV Jornadas de Computación Reconfigurable y Aplicaciones, Bellaterra (Spain) (in Spanish) (Sep 2004)
5. Chapell, S., Sullivan, C.: Handel-C for co-processing an co-design of field programmable systems on chip. Proceeding of the JCRA'02 (2002)
6. Claver, J.M., Carrión, M.C., Canseco, M., Caminero, M.B., Quiles, F.J.: A New Hardware Efficient Link Scheduling Algorithm to Guarantee QoS on Clusters. Lecture Notes in Computer Science, Euro-Par 2005 Parallel Processing: 11th International, Ed. Springer-Verlag (2005)
7. Celoxica: RC1000 Software reference manual. (2001)
8. Gaughan, P.T., Yalamanchili, S.: Adaptive routing protocols for direct multiprocessor networks. IEEE Computer (May 1993)
9. Krishnamurthy, R.,Yalamanchili, S., Schwan, K., and West, R.: Architecture and Hardware for Scheduling Gigabit Packet Streams. Proceedings of the 10th Symposium on High Performance Interconnects Hot Interconnects (HotI'02) (2002)
10. Martínez, A., Alfaro, F.J., Sánchez, J.L., Duato, J.: Providing Full QoS Support in Clusters Using Only Two VCs at the Switches. XII IEEE International Conference on High Performance Computing (HiPC), Goa (India) (Dec 2005)

11. Rexford, J.L., Greenberg, A.G., Bonomi, F.G.: Hardware-efficient fair queuing architectures for High speed networks. Proceedings of the IEEE INFOCOM '96, San Francisco (Mar 1996).
12. Shanley, T.: Infiniband Network Architecture. Addison-Wesley (2003)
13. Moon, S., Rexford, J., Shin, K.: Scalable hardware priority queue architectures for high speed packet switches. IEEE Trans. on Computers, V. 49 No. 11, pp. 1215-1227 (2000)
14. West, R. and Poellabauer, C.: Analysis of a Window-Constrained Scheduler for Real-time and Best-Effort Packet Streams. Proceedings of the 21st Real Time Systems Symposium (Nov 2000)

Studying Several Proposals for the Adaptation of the DTable Scheduler to Advanced Switching*

Raúl Martínez, Francisco J. Alfaro, and José L. Sánchez

Departamento de Sistemas Informáticos, Universidad de Castilla-La Mancha
02071 - Albacete, Spain
{raulmm, falfaro, jsanchez}@info-ab.uclm.es

Abstract. Advanced Switching (AS) is a new fabric-interconnect technology that further enhances the capabilities of PCI Express. On the other hand, the provision of Quality of Service (QoS) in computing and communication environments is currently the focus of much discussion and research in industry and academia.

A key component for networks with QoS support is the egress link scheduling algorithm. AS defines a table-based scheduler that is simple to implement and can offer good latency bounds with a fixed packet size. However, it does not work properly with variable packet sizes and faces the problem of bounding the bandwidth and latency assignments.

In this paper we propose several possible modifications to the original AS table scheduler in order to implement the Deficit Table (DTable) scheduler. This scheduler works properly with variable packet sizes and allows to partially decouple the bandwidth and latency assignments.

1 Introduction

Advanced Switching (AS) [1] is a new high performance interconnection technology based on PCI Express [10],which is the next generation of the PCI bus.Whereas PCI Express has already begun to reshape a new generation of PCs and traditional servers, AS is intended to proliferate in multiprocessor, peer to peer systems in the communications, storage, networking, servers and embedded platform environments. Together, PCI Express and AS have the potential for building the next generation interconnects [8].

The provision of Quality of Service (QoS) in the environment where AS is foreseen to be used will be very important. A key component for networks with QoS support is the egress link scheduling algorithm.AS defines two egress link schedulers: The virtual channel arbitration table scheduler and the Minimum Bandwidth egress link scheduler (MinBW). The table-based scheduler is simple

* This work was partly supported by the Spanish CICYT under Grant TIC2003-08154-C06-02, by the Junta de Comunidades de Castilla-La Mancha under Grant PBC-05-005-1, and by the Spanish State Secretariat of Education and Universities under FPU grant.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 98–112, 2006.
© Springer-Verlag Berlin Heidelberg 2006

to implement and can offer good latency bounds with a fixed packet size. However, it does not work properly with variable packet sizes and faces the problem of bounding the bandwidth and latency assignments [7].

In [4] we proposed a new table-based scheduler, which we called Deficit Table (DTable), which works properly with variable packet sizes. We also proposed a methodology to configure this scheduler that allows us to decouple, at least partially, the bandwidth and latency assignments.

In [7] we examined the AS mechanisms intended for providing QoS and showed how to provide QoS based on bandwidth and latency requirements. Specifically, in [7] we showed how to use a limited version of the DTable scheduler to implement the AS table scheduler in order to support variable packet sizes. The original DTable scheduler considers a different weight per table entry, however, in this version we used the same weight in all the table entries. The resulting scheduling mechanism does not require to modify the interface provided in the AS specification for configuring the table scheduler and only requires simple hardware modifications of the original AS table scheduler. However, if we want to take advantage of the decoupling configuration methodology, we need to be able to assign different weights to the table entries.

In this paper we propose three different ways of implementing a full version of the DTable scheduler to substitute the original AS table scheduler, but modifying as little as possible the AS specification. Moreover, we review the methods we have proposed to provide QoS over AS, now employing a full version of the DTable scheduler. Finally, we evaluate the performance of our proposals by simulation. comparing the performance of the DTable scheduler with the performance of the MinBW scheduler.

The structure of the paper is as follows: Section 2 presents a summary of the general aspects in the AS specification including the most important mechanisms that AS provides to support QoS. Section 3, reviews the DTable scheduler and our methodology to decouple the bandwidth and latency assignments. In Section 4, we propose how to fully implement the DTable scheduler in AS. In Section 5 we propose how to provide QoS requirements over AS using a full implementation of the DTable scheduler. Details on the experimental platform and the performance evaluation are presented in Section 6. Finally, some conclusions are given.

2 Advanced Switching Revision

Advanced Switching (AS) is built on the same physical and link layers as PCI Express. Moreover, it includes an optimized transaction layer, providing a rich set of features and capabilities. A credit-based flow control protocol ensures that packets are only transmitted when there is enough buffer space at the other end to store them, making sure that no packets are dropped when congestion appears. The flow control credit unit is 64 bytes.

An AS fabric permits us to employ Virtual Channels (VCs), egress link scheduling, and an admission control mechanism to differentiate between traffic flows. AS uses VCs to aggregate flows with similar characteristics. AS supports

up to 20 VCs of three different types: Up to 8 bypassable unicast VCs, up to 8 ordered-only unicast VCs, and up to 4 multicast VCs. The bypassable VC with the highest identifier in each network element is called the Fabric Management Channel (FMC). Note that each VC has its own credit count for the credit-based flow control. Moreover, each VC type has its own MTU. The allowed MTU values for the bypassable VC type are 192, 320, 576, 1088, and 2176 bytes. The allowed MTU values for the ordered VC type are 64, 96, 128, 192, 320, 576, 1088, and 2176 bytes.

In AS, the arbitration is made at VC level. AS defines two schedulers to resolve between the up to twenty VCs competing for bandwidth onto the egress link: The table scheduler and the MinBW scheduler.

The table scheduler employs an arbitration table that consists in a register array with fixed-size entries of 8 bits. Each entry contains a field of 5 bits with a VC identifier value and a reserved field of 3 bits. When arbitration is needed, the table is cycled through sequentially and a packet is transmitted from the VC indicated in the current table entry regardless of the packet size. If the current entry points to an empty VC, that entry is skipped. The number of entries may be 32, 64, 128, 256, 512, or 1024.

The MinBW scheduler is intended for a more precise allocation of bandwidth regardless of the packet size. This scheduler consists of two parts: The first is a mechanism to provide the FMC with absolute priority, ahead of the other VCs, but with its bandwidth limited by a token bucket. The second is a mechanism to distribute bandwidth amongst the rest of the VCs according to a configurable set of weights. In [5] we proposed several implementations for this scheduler that accomplish all the properties that the AS specification indicates [1], including the interaction with the AS flow control.

3 The Deficit Table Scheduler

The Deficit Table (DTable) scheduler [4], defines an arbitration table in which each table entry has associated a flow[1] identifier and an *entry weight*, which is usually expressed in flow control credits in networks with a credit-based link-level flow control. Moreover, each flow has assigned a *deficit counter* that is set to 0 at the beginning. When scheduling is needed, the table is cycled through sequentially until an entry assigned to an active flow is found. A flow is considered active when it stores at least one packet and the flow control allows that flow to transmit packets. When a table entry is selected, the *accumulated weight* is computed. The accumulated weight is equal to the sum of the deficit counter for the selected flow and the current entry weight. The scheduler transmits as many packets from the active flow as the accumulated weight allows. When a packet is transmitted, the accumulated weight is reduced by the packet size.

The next active table entry is selected if the flow becomes inactive or the accumulated weight becomes smaller than the size of the packet at the head of

[1] In this paper we will use the term *flow* to refer both to a single flow or to an aggregated of several flows with similar characteristics.

the queue. In the first case, the remaining accumulated weight is discarded and the deficit counter is set to zero. In the second case, the unused accumulated weight is saved in the deficit counter, representing the amount of weight that the scheduler owes the queue.

In [4] we also proposed a methodology to configure the DTable scheduler to decouple, at least partially, the bounding between the bandwidth and latency assignments. With this methodology we set the maximum distance between any consecutive pair of entries assigned to a flow depending on its latency requirement. Moreover, we can assign the flows with a bandwidth that depends on the total weight assigned to the flow entries. The proportion of the egress link bandwidth that can be actually assigned to a flow depends not only on the proportion of table entries assigned, but also on two table configuration parameters. We have called these parameters w and k.

The w parameter determines the maximum weight M that can be assigned to a single table entry in function of the General MTU of the network ($GMTU$): $M = GMTU \times w$. The k parameter determines the total weight that can be distributed between all the table entries. We call this value the *bandwidth pool*: $pool = N \times GMTU \times k$.

The minimum weight that a table entry can have associated should ensure that there will never be necessary to cycle through the entire table several times in order to gather enough weight for the transmission of a single packet. Therefore, in [4] we set this minimum value to the GMTU. However, in [6] we proposed to use different MTUs for the different flows. This means that each flow has a specific MTU equal to or lower than the GMTU. Therefore, we can assign a table entry with a minimum weight equal to its flow's specific MTU instead of the GMTU. Using different MTUs for the different flows allows us to assign a smaller amount of bandwidth to those flows with a smaller MTU than the general MTU. Moreover, the value of the k parameter can be smaller, and thus the flexibility to assign the bandwidth increases. However, the specific flow MTU must be assigned taking into account the characteristics of the traffic flow. A too small specific MTU may decrease the latency performance of the flow [6].

Therefore, supposing an arbitration table with N entries, the minimum bandwidth $min\phi_i$ and the maximum bandwidth $max\phi_i$ that can be assigned to the i^{th} flow are:

$$min\phi_i = \frac{n_i \times MTU_i}{pool} = \frac{n_i \times MTU_i}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{MTU_i}{GMTU} \times \frac{1}{k} \quad (1)$$

$$max\phi_i = \frac{n_i \times M}{pool} = \frac{n_i \times GMTU \times w}{N \times GMTU \times k} = \frac{n_i}{N} \times \frac{w}{k} \quad (2)$$

Note that varying the w and k parameters affects the minimum and maximum bandwidth that can be assigned to all the flows. However, assigning a specific MTU to a flow only affects the minimum bandwidth of that flow.

4 Implementing the DTable in AS

As stated before, the AS arbitration table consists in a list of VC identifiers without any weight assigned to each entry as it is the case in the DTable scheduler. In [7] we proposed to convert the AS table scheduler into the DTable scheduler assigning each table entry with the same weight, the MTU. Moreover, we internally assign each VC with a deficit counter. These little changes require simple hardware modifications of the original AS table scheduler but they do not require to modify the interface provided in the AS specification for configuring the table scheduler. Note that these counters are set to zero at the beginning and are modified dinamically by the scheduler itself during the scheduling process, and thus they do not require any user configuration.

These simple modifications solve the problem of the AS table scheduler with variable packet sizes but it does not allow to decouple the bandwidth and the latency assignments using our configuration methodology because this methodology relies on using different weights for the table entries.

In this paper we propose several possibilities to fully implement the DTable scheduler in AS. Our objective is to be able to assign the table entries with different weights but modifying as little as possible the AS specification. We propose three possibilities to assign each table entry with a weight: to use the 3-bit reserved field of each table entry, to modify the arbitration table structure, and to use the same weight for all the entries of a VC.

4.1 Using the 3-bit Reserved Field

The easiest possibility to implement the DTable scheduler in AS is to employ the 3-bit reserved field to assign a weight to each entry. The problem of this implementation is that this field only allows us to specify a weight between 0 and 7, and thus, several considerations must be made. First of all, as stated before, the entry weight must represent at least the value of the GMTU. Therefore, a weight of 0 is not going to be used, and thus, we propose to consider the weight 0 as 1, the weight 1 as 2, etc. This allows us to specify a weight between 1 and 8 with the 3-bit field.

Moreover, in AS, the GMTU can be up to 34 flow control credits (2176 bytes). Obviously, it is not possible to represent directly a value of at least 34 with just 3 bits. Therefore, when using the 3-bit reserved field to assign a weight to each entry, each weight unit will represent a weight equivalent to a certain number of flow control credits. The maximum weight per entry, the bandwidth pool, and the actual value assigned to each table entry are expressed in weight units. Moreover, in order to calculate the minimum and maximum bandwidth that can be assigned to a VC, we must have a value in weight units for the GMTU and the specific MTUs, if applicable. However, the scheduler is still going to consider flow control units to arbitrate. This means that the deficit counter and the accumulated weight are still going to be expressed in flow control credits. Therefore, when an entry is selected its weight must be translated into its value in flow control credits. $accumulated\_weight = deficit\_counter + ((entry\_weight +$

1) × *credits_per_weight_unit*). When configuring the DTable scheduler we must specify, apart from the VC identifier and weight of each table entry, the number of flow control credits that represents each weight unit.

This implementation posibility limits the maximum weight per entry to 8, and thus the maximum value for the w parameter is also limited to 8 (with the minimum possible value of the GMTU: 1). The values of the GMTU and the specific MTUs are also very limited (1-8). The bandwidth assignation granularity depends on the bandwidth pool. The maximum bandwidth pool is the maximum weight per table entry multiplied by the number of table entries, and thus, the maximum granularity is $1/(8\times N)$.

Summing up, this possibility limits the possible values for the w parameter and the specific MTUs. This limits the flexibility of the table configuration. However, the implementation of this option is quite simple.

4.2 Modifying the Arbitration Table Format

Other possibility is to modify the structure of the arbitration table in order to dedicate a higher number of bits to the entry weight. Specifically, we propose to use two bytes per table entry, and use 5 bits for the VC identifier and up to 11 for the entry weight. With 11 bits, the entry weight can take a value between 1 and 2048, and thus, with a MTU of 34 flow control crdits, the maximum w parameter is around 60 ($M = GMTU \times w$, $w = 2048/34$) and the maximum granularity is $1/(N\times 2048)$

This possibility allows a higher flexibility in the assignation of the w parameter and the specific MTUs values. However, it requires the double of memory to store the arbitration table than the previous option for the same number of entries. Moreover, it requires processing two bytes per entry instead of only one.

4.3 Using Only One Weight Per VC

The third possibility that we propose is to associate the same weight to all the entries assigned to a VC. Therefore, we only need to specify a table weight per VC instead of per table entry. In order to change as little as possible the AS specification a possibility is to specify the weight assigned to the entries of each VC employing the MinBW configuration structure, which provides 12 bits to specify a weight per each VC. This allows us to specify a weight between 1 and 4096, and thus, the maximum w value is around 120 ($M = GMTU \times w$, $w = 4096/34$). When a new table entry is selected, the accumulated weight is computed as: $accumulated\_weight = deficit\_counter + VC\_weight$.

This possibility also allows us a higher flexibility in the assignation of the w parameter and the specific MTU values than the 3-bit option. The disadvantage of this possibility is that we cannot assign the weight units from the bandwidth pool in a totally free way between the table entries. We have to assign the weights in exact fractions of the number of entries of each VC. Therefore, the bandwidth assignation granularity is different for each VC and depends on the number of entries assigned to that VC: $n_i/(N\times 4096)$.

5 Providing QoS over AS with the DTable Scheduler

In [7] we examined the AS mechanisms and showed how to provide QoS to the applications. First of all, a set of Service Classes (SCs) with different requirements must be specified. The egress link scheduler must be properly configured to provide the different SCs with their requirements. Moreover, an admission control protocol must be used to provide QoS guarantees.

In order to define this set of SCs, we propose a traffic classification based on two network parameters: Bandwidth and latency. We distinguish three broad categories of traffic: Network control traffic, QoS traffic, and best-effort traffic. The network control traffic is high-priority traffic used to maintain and support the network infrastructure. The QoS traffic has explicit minimum bandwidth and/or maximum latency requirements. The best-effort traffic is largely insensitive to both bandwidth and latency and is only characterized by the differing priority among each other.

When various flows obtain access to the AS fabric, they will be aggregated into the SCs depending on their characteristics. If there are sufficient VCs, we will devote a separate VC to each SC. The control SC will be assigned to the FMC. Note, however, that this VC does not have maximum priority when using the table scheduler, so we will consider it as any other VC with traffic of high latency requirements.

In order to provide QoS guarantees, an Admission Control (AC) mechanism must be used for the QoS SCs. Note that this is not necesary for the control and best-effort SCs. The AC mechanism would allow a new QoS connection to be established if there is enough bandwidth all along its path.

As stated before, when using the DTable scheduler, in order to provide maximum latency requirements to the traffic of a VC, the maximum separation between two consecutive table entries devoted to that VC must be fixed. In order to provide traffic of a given VC with a minimum bandwidth, the amount of weight units from the bandwidth pool assigned to those VC table entries must accomplish with the proportion of desired egress link bandwidth. Therefore, when we know the maximum distance between two consecutive table entries, and thus, the number of entries, and the amount of bandwidth that we want to assign to each VC, we must choose the w and k parameters that make possible this distribution.

Moreover, we can limit the MTU of some VCs in order to have a smaller minimum bandwidth for those VCs and for being able to use smaller k values. We can assign each VC a different MTU at a communication library level, but this would entail to add complexity to the AS communication protocols. On the other hand, we can take advantage of the AS characteristics to simplify the process. As stated before, AS allows us to establish two different MTUs for the two unicast VC types. Therefore, we can have two sets of VCs with two different MTUs and we can assign the SCs to the VCs taking into account this. Note that those SCs that have high latency requirements, and thus require more table entries, usually have small bandwidth requirements and use small packets. Therefore, we can assign these SCs to the VCs with the smallest MTU.

6 Performance Evaluation

In this section, we evaluate the performance of our proposals to provide latency and bandwidth requirements over AS with a full implementation of the DTable scheduler by simulation. We consider a realistic multimedia scenario with different types of traffic. This allows us to apply our proposals of using different MTUs taking into account the traffic characteristics. Note, however, that our proposals are equally valid for other environments. We also compare the flexibility in the bandwidth assignation of the three possibilities that we propose to implement the DTable scheduler.

Moreover, we compare the performance of the DTable scheduler with the performance of the MinBW scheduler implemented with the Deficit Round Robin (DRR) [11] and the Self-Clocked Weighted Fair Queuing (SCFQ) [2] algorithms. We have employed the *credit aware* versions of this schedulers that we proposed to be used for the implementation of the MinBW [5]. We have chosen the SCFQ-CA algorithm as an example of "sorted-priority" algorithm with a good latency performance and the DRR-CA algorithm because of its very low computational complexity. In order to simplify the comparison among the three schedulers, we have not reflected the different complexity into different scheduling time.

6.1 Simulated Architecture

We have used a perfect-shuffle Bidirectional Multistage Interconnection Network (BMIN) with 64 end-points connected using 48 8-port switches (3 stages of 16 switches). In AS any topology is possible, but we have used a MIN because it is a common solution for interconnection in current high-performance environments. The switch model uses a combined input-output buffer architecture with a crossbar to connect the buffers. Virtual output queuing has been implemented to solve the head-of-line blocking problem at switch level.

In our tests, the link bandwidth is 2.5 Gb/s but, with the AS 8b/10b encoding scheme, the maximum effective bandwidth for data traffic is only 2 Gb/s. We are assuming some internal speed-up (x1.5) for the crossbar, as is usually the case in most commercial switches. AS gives us the freedom to use any algorithm to schedule the crossbar, so we have implemented a round-robin scheduler. The time that a packet header takes to cross the switch without any load is 145 ns, which is based on the unloaded cut-through latency of the AS StarGen's *Merlin* switch [12]. We are going to use the maximum MTU allowed in AS: 2176 bytes (34 flow control credits). The buffer capacity is 17408 bytes (8×MTU) per VC both at the input and at the output ports of the switches.

6.2 Simulated Scenario

The IEEE standard 802.1D-2004 [3] defines 7 traffic types at the Annex G, which are appropriate for this study. We will consider each traffic type as a SC. Table 1 shows each SC and its requirements. In this way, the workload is composed of

Table 1. SCs suggested by the standard IEEE 802.1D-2004

Type	SC	Description
Control	Network control (NC)	Traffic to support the network infrastructure.
QoS	Voice (VO)	Traffic with a limit of 10 ms for latency and jitter.
QoS	Video (VI)	Traffic with a limit of 100 ms for latency and jitter.
QoS	Controlled load (CL)	Traffic with explicit bandwidth requirements.
Best-effort	Excellent-effort (EE)	Preferential best-effort traffic.
Best-effort	Best-effort (BE)	LAN traffic as we know it today.
Best-effort	Background (BK)	Traffic that should not impact other flows.

7 SCs and each one of them will be assigned to a different VC. As stated before, the NC SC is assigned to the FMC.

We suppose a scenario in which the goal is to dedicate around 20-25% of the egress link bandwidth to best-effort traffic, around 5-10% bandwidth to voice traffic (a lot but low-bandwidth requiring connections), around 5-10% of bandwidth to controlled load, and 40-50% of bandwidth to video traffic (a lot and high-bandwidth requiring connections). This percentages are intended to represent a multimedia scenario with a realistic combination of traffic from applications with very different requirements. Moreover, we expect that the maximum network throughput to be around 85-95%. We also consider that control and voice traffic uses small packet sizes, as it is usually the case. In [13] several payload values for voice codec algorithms are shown. These values range from 20 bytes to 160 bytes. We have selected a payload of 160 bytes for voice traffic.

6.3 Scheduler Configuration

The table scheduler must be properly configured to provide the SCs with their bandwidth and latency requirements. A table of 128 entries has been used. Table 2 shows the distribution of the table entries among the SCs. It shows the maximum distance between any consecutive pair of entries, the number of table entries, and the percentage of entries that this entails for each SC. We have assigned a distance of 2, 4, and 8 to the NC, VO, and VI SCs respectively, attending to their latency requirements. Note that this entails assigning 112 entries. We have distributed 8 entries among the best-effort SCs attending to the different priority among them. Finally, we have assigned the remaining 8 entries to the CL SC. For the CL SC and the best-effort SCs we could have assigned the entries sequentially in the free gaps of the table, but to achieve better latency results for these SCs we have assigned their entries minimizing the distance between any pair of consecutive entries.

Table 2 also shows the weight configuration that we have chosen to fit the bandwidth requirements stated in the previous section using the 3-bit implementation option of the DTable scheduler. As stated before, we must choose a value between 1 and 8 to represent the GMTU (in this case 34 flow control credits). We have chosen to represent the GMTU using 3 weight units. Therefore, each weight unit represents 12 flow control credits and the w value is 2.67. Moreover, we are going

Table 2. Table entries distribution and weight configuration with the 3-bit option. $w = 2.67$, $M = 8$, $k = 0.76$, $pool$=292.

	Table entries distribution			Weight configuration (3-bit option)				
VC	**Max. distance**	**#entries**	**%entries**	**MTU**	$min\phi_i$	$max\phi_i$	ϕ_i	**Total weight**
NC	2	64	50	1	21.92	175.34	28.08	82
VO	4	32	25	1	10.96	87.67	10.96	32
VI	8	16	12.5	3	16.44	43.84	43.84	128
CL	16	8	6.25	3	8.22	21.92	8.91	26
EE	32	5	3.91	3	5.14	13.70	5.14	15
BE	64	2	1.56	3	2.05	5.48	2.05	6
BK	128	1	0.78	3	1.03	2.74	1.02	3
	Total	128	100	-	65.76	350.69	100	292

to use a smaller MTU for the NC and VO SCs. Specifically, we set a specific MTU of 1 weight unit, which actually entails a specific MTU of 576 bytes. Note that assigning a smaller MTU to these SCs is not going to decrease their performance because the control and voice traffic already use small packets.

Table 2 shows the minimum and maximum bandwidth that this configuration entails (according to expressions 1 and 2) and the bandwidth ϕ_i that has been finally assigned to each SC. Moreover, it shows the total number of weight units from the bandwidth pool that each SC has been assigned to their entries. In order to choose these parameters we have considered mainly how to provide the VO SC with a small amount of bandwidth at the same time that we assign the VI SC a high amount of bandwidth.

Table 3 shows the weight configuration of the other two options that we propose as possible implementations of the DTable scheduler: to modify the structure of the arbitration table to use more bits for the weight and to use only one weight per VC. In order to compare these two cases with the 3-bit option, we have used the same values for the w and k parameters. Note, however, that the value of w is 2.68 instead of 2.67 because the value of M must be an integer

Table 3. Weight configuration with the modification of the table structure and the weight per VC options. $w = 2.68$, $M = 91$, $k = 0.76$, $pool$=3308.

				Modifying the table		One weight per VC	
VC	**MTU**	$min\phi_i$	$max\phi_i$	ϕ_i	**Total weight**	ϕ_i	**Total weight**
NC	9/5	17.41/9.67	176.06	28.08	928	27.09/29.02	896/960
VO	9/5	8.71/4.84	88.03	10.96	363	10.64/11.61	352/384
VI	34	16.45	44.01	43.84	1450	43.53/44.01	1440/1456
CL	34	8.22	22.01	8.91	295	8.71/8.95	288/296
EE	34	5.14	13.75	5.14	170	5.14	170
BE	34	2.05	5.50	2.05	68	2.05	68
BK	34	1.03	2.75	1.03	34	1.03	34
	Total	59.01/47.4	352.11	100	3308	98.19-101.81	3248-3368

Table 4. Injected traffic and scheduler configuration

SC	Injection rate	Traffic pattern	Packet size (bytes)
NC	0.01	Bursts60	up to 576
VO	0.119	64 Kb/s CBR	168
VI	0.438	750 Kb/s MPEG-4 traces	up to 2176
CL	0.089	750 Kb/s CBR	2176
EE	$0 \rightarrow 0.115$	Bursts60	up to 2176
BE	$0 \rightarrow 0.115$	Bursts60	up to 2176
BK	$0 \rightarrow 0.115$	Bursts60	up to 2176
Total	$0.656 \rightarrow 1.001$		

($M = GMTU \times w$, $91 = 34 \times 2.68$). These two options have the same minimum and maximum bandwidth per VC. Note that the values are slightly different from the 3-bit option because in these two cases the MTUs are expressed in flow control credits and not in weight units, which is an approximation of the real value. Note also that contrary to the 3-bit case, in these cases we could have assigned a smaller specific MTU to have smaller minimum bandwidths for the NC and VO SCs. Table 3 shows an example with a MTU of 320 bytes (5 flow control credits). The main difference between the options of modifying the table structure and using one weight per VC is the granularity in the bandwidth assignment. Table 3 shows that with one weight per VC the same minimum bandwidth values that in the 3-bit case cannot be assigned for all the VCs. This table shows the two nearest values that could be assigned instead. Note that the granularity depends on the number of table entries.

Note that the main difference of the three DTable implementations is the way of specifying the weight of a given table entry. This is going to affect the flexibility to assign the bandwdith to the VCs, but not the performance that they provide. Moreover, the three implementations probably differ also in their computational complexity. However, for the sake of simplicity we have considered the same computaion time for all of them. Therefore, we only show the performance results obtained with the 3-bit implementation. In order to compare the performance of the DTable and MinBW schedulers, both proposed implementations of the MinBW scheduler (DRR-CA and SCFQ-CA) have been configured to provide a minimum bandwidth equal to the stated for the DTable case in Table 2.

6.4 Traffic Pattern

The packets are generated according to different distributions, as can be seen in Table 4. VO, VI, and CL SCs are composed of point-to-point connections of the given bandwidth. VO and CL SCs are generated following a Constant Bit Rate (CBR) distribution. In the case of the VI SC, a video trace is used to generate the size of each frame. Each frame is injected into the network interfaces every 40 ms. If the frame size is bigger than the MTU, the frame is split into several packets. The traffic of the NC, EE, BE, and BK SCs is generated according to a Bursts60 distribution [9], which models worst-case real traffic scenarios.

Our intention is to show that with an AC mechanisms for controlling the QoS traffic and a relatively small amount of control traffic (as it is usually the case), the QoS requirements of the different SCs are met, whatever the load of best-effort traffic. For that purpose, we inject a fixed amount of traffic of each QoS SC (VO, VI, and CL) equal to the minimum bandwidth that they have been assigned, representing the maximum injection allowed by the AC mechanism. Moreover, we inject a fixed amount of control traffic (NC), and gradually increase the amount of best-effort traffic (EE, BE, and BK). Table 4 shows the proportion of traffic of each SC that each node injects regarding the link bandwidth. The destination pattern is uniform in order to fully load the network.

6.5 Simulation Results

Figure 1 shows throughput and latency performance per SC of the DTable schedulers. The bandwidth performance for the MinBW schedulers is the same and the average and maximum latency performance follow the same trends. Regarding the throughput performance, Figure 1 shows that the NC and the QoS SCs obtain all the bandwidth that they inject. However, when the network load is high (around 85%-90%), the best-effort SCs do not yield a corresponding result. From that input load, these SCs obtain a bandwidth proportional to their priority. Regarding the latency performance, Figure 1 shows that the average and maximum latency of the NC SC and the QoS SCs grow with the load until they reach a certain value. Once this value is reached the latency remains more or less constant. On the other hand, the average and maximum latency of best-effort SCs grow with the load. Furthermore, it can be seen that best-effort SCs obtain a different average and maximum latency according to their priority.

As stated before the three schedulers offer the same throughput performance but differ on the latency performance. Figure 2 compares the latency performance of the NC and QoS SCs of the three schedulers. Specifically, it shows the percentage of improvement on average and maximum latency of the DTable scheduler over the MinBW scheduler implemented with the SCFQ-CA algorithm and the MinBW scheduler implemented with the DRR-CA algorithm.

Figure 2 shows that both implementations for the MinBW scheduler obtain a better latency performance for the NC SC. This is because this SC is assigned to the FMC VC, which has maximum priority over the rest of VCs with the MinBW scheduler. However, we can say that the DTable scheduler obtains a very good performance if we consider that with this scheduler the FMC is treated like any other VC. Note that with this scheduler the average latency is only around 20% worse and the maximum latency around 10% worse than with the MinBW scheduler. This is because we have assigned a maximum distance between any consecutive pair of entries of 2.

Regarding the QoS SCs Figure 2 shows that the DTable scheduler provides a quite better latency performance than the DRR-CA scheduler for the VO SC and a slightly worse latency than the SCFQ-CA scheduler. Again this difference is smaller for the maximum latency statistic. Moreover, this figure shows that the improvement of the DTable scheduler over the DRR-CA scheduler for the VI SC

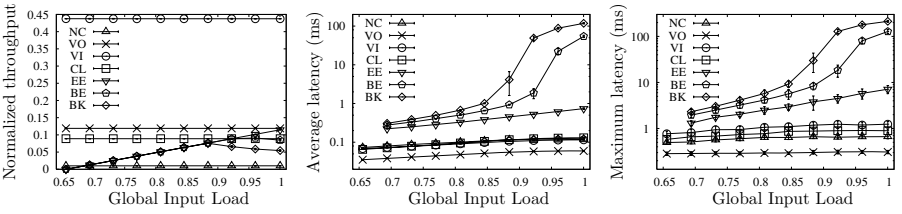


Fig. 1. Throughput and latency performance per SC of the DTable schedulers

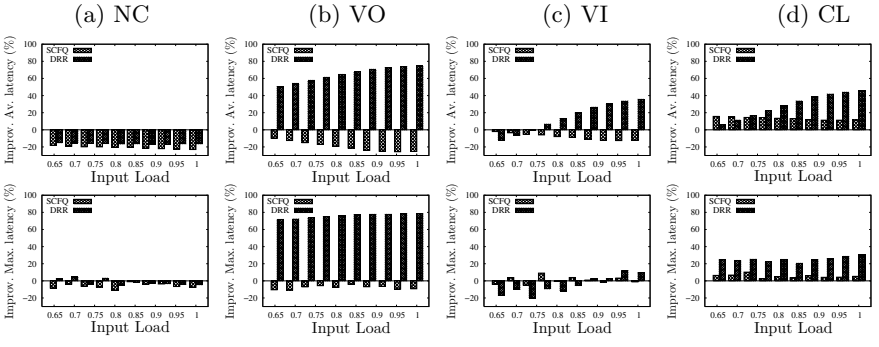


Fig. 2. Average and maximum latency improvement of the DTable scheduler over the MinBW scheduler

depends in high degree on the network load. When the load is low, the average latency performance provided by the DTable scheduler is slightly worse than the latency provided by the DRR-CA. However, the DTable scheduler provides a better latency when the load is high. The SCFQ-CA scheduler provides in any case a slightly better average latency for this SC. The maximum latency performance of the VI SC presents a similar behaviour, however, the variability of the maximum latencies is higher and the differences among the three schedulers are not so clear. Regarding the CL SC, the DTable scheduler provides a better average and maximum latency performance than both MinBW implementations.

7 Conclusions

AS provides a table-based scheduler that does not work properly with variable packet sizes and faces the problem of bounding the bandwidth and latency assignments. In this paper we propose three possible modifications to the original AS table scheduler in order to fully implement the DTable scheduler. The objective of these modifications has been to be able to assign each table entry with a weight, but modifying as little as possible the AS specification.

Using the 3-bit reserved field is probably the simplest possibility. However, it limits the possible values for the w parameter and the specific MTUs. The possibility of using the same weight for all the entries of a VC allows us to use

higher values for the w parameter and to choose freely the values for the specific MTUs. However, the bandwidth assignation granularity is different for each VC and depends on the number of entries assigned to that VC. The possibility of modifying the arbitration table structure does not present these problems, however, it requires a higher amount of memory to store the arbitration table and needs to process two bytes, instead of one, per table entry.

We have evaluated our proposals in a multimedia scenario and have compared the performance of the DTable scheduler with two different implementations for the MinBW scheduler: the SCFQ-CA and the DRR-CA algorithms. The simulation results show that all the schedulers provide the same bandwidth performance, but a different latency performance. The DTable scheduler provides a better latency performance than the DRR-CA scheduler and only slightly worse than the SCFQ-CA scheduler, which has a complexity that makes it difficult to be implemented in a high performance network, with only a slightly higher complexity than in the DRR-CA case.

Summing up, we have shown that modifying slightly the AS specification it is possible to solve in a high degree the problems of the original AS table scheduler. The resulting scheduler would allow us to provide a good latency performance with a small computational complexity.

References

1. Advanced Switching Interconnect Special Interest Group. *Advanced Switching core architecture specification. Revision 1.0*, December 2003.
2. S. J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *INFOCOM*, 1994.
3. IEEE. 802.1D-2004: Standard for local and metropolitan area networks. `http://grouper.ieee.org/groups/802/1/`, 2004.
4. R. Martínez, F.J. Alfaro, and J.L. Sánchez. Decoupling the bandwidth and latency bounding for table-based schedulers. *International Conference on Parallel Procesing (ICPP)*, August 2006.
5. R. Martínez, F.J. Alfaro, and J.L. Sánchez. Implementing the advanced switching minimum bandwidth egress link scheduler. *IEEE International Symposium on Network Computing and Applications (IEEE NCA06)*, July 2006.
6. R. Martínez, F.J. Alfaro, and J.L. Sánchez. Improving the flexibility of the deficit table scheduler. Technical Report DIAB-06-06-1, Departamento de Informática Universidad de Castilla-La Mancha, June 2006. Also submitted to the International Conference on High Performance Computing (HiPC) 2006. Available at `https://www.info-ab.uclm.es/trep.php`.
7. R. Martínez, F.J. Alfaro, and J.L. Sánchez. Providing Quality of Service over Advanced Switching. *International Conference on Parallel and Distributed Systems (ICPADS)*, July 2006.
8. D. Mayhew and V. Krishnan. PCI Express and Advanced Switching: Evolutionary path to building next generation interconnects. In *Hot Interconnects: 10th Symposium on High Performance Interconnects*, 2003.
9. M. Katevenis N. Chrysos. Multiple priorities in a two-lane buffered crossbar. In *Proceedings of the IEEE Globecom 2004 Conference*, November 2004.

10. PCI SIG. *PCI Express base architecture specification. Revision 1.0a*, April 2003.
11. M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *SIGCOMM*, pages 231–242, 1995.
12. StarGen. *StarGen's Merlin switch*, 2004. `http://www.stargen.com/products/merlin_switch.shtml`.
13. A. Tyagi, J. K. Muppala, and H. de Meer. VoIP support on differentiated services using expedited forwarding. In *IEEE International Performance, Computing, and Communications Conference (IPCCC)*, February 2000.

Asymmetrical SSL Tunnel Based VPN*

Jingli Zhou, Hongtao Xia, Jifeng Yu, and Xiaofeng Wang

Huazhong University of Science and Technology, Wuhan, 430074, China
{jlzhou,htxia,jfyu,xfwang}@wtwh.com.cn

Abstract. Asymmetric SSL Tunnel (AST) based Virtual Private Network is presented as a cheap solution for large scale SSL VPNs. In this solution, portion of SSL/TLS computational load is transferred to disengaged internal application servers, so that VPN server is no more the bottleneck of VPN system. This paper analyzes the performance advantage of asymmetric SSL tunnel over traditional SSL tunnel, and discusses the secret management scheme for AST, which can meet enhanced security requirement and synchronize cipher specs of multipoint. Finally, a kernel optimization algorithm was introduced. AST is implemented in OpenVPN, which is originally a stable traditional SSL VPN solution. Experiment shows that the overall throughput of OpenVPN can be greatly improved after AST adopted.

1 Introduction

SSL VPN is a promising secure remote access solution [1]. It is based on SSL/TLS [2,3] protocol, and thus has the following outstanding advantages: low cost, easy-to-deploy, fine-grained access control, and etc. But its performance and scalability are also hampered by the computation overhead of SSL/TLS protocol [4,5,6].

SSL tunnel is overlay network facility for creating a SSL VPN on top of existing Internet or IP based network. In SSL tunnel, the payload of an IP packet carries another full IP packet including its header information. That IP packet to be carried is compressed and encrypted in advance according to SSL/TLS protocol, see figure 1.

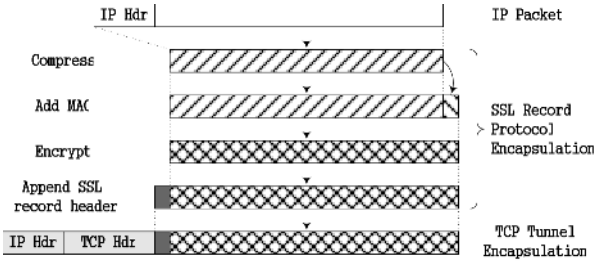


Fig. 1. An IP Packet is SSL encapsulated, and carried by another IP packet of SSL tunnel

* This work was supported by NSFC (No. 60373088).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 113 – 124, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The computation overhead introduced by SSL/TLS protocol can be divided into two parts. When creating a SSL tunnel, the SSL Handshake Protocol was used by communication peers to authenticate each other, and to negotiate encryption and message authentication coding (MAC) algorithms, coupled with cryptographic keys. Those algorithms and keys are used to protect data in subsequent communication, and called as **cipher spec**. The computational load of negotiating a new cipher spec is called handshake overhead. Handshake overhead only occurs once when VPN client logon, when VPN client connects specific internal server, cipher spec can be reused.

After that, application data can be transferred. Both peers use the cipher spec to do SSL encapsulation or decapsulation, according to SSL Record Protocol. As figure 1 shows, for each outgoing packet, the sender peer needs to do compressing, computing MAC, and encrypting. The reversed processes are needed in the receiver peer: decrypting, verify MAC and decompressing. The computational loads of both peers are approximately the same, and are called as transfer overheads.

In traditional VPN solutions, VPN server is the common end of all SSL tunnels. As figure 2 shows, VPN server was always deployed in front of a LAN, and acts as a portal. SSL tunnels are created connecting VPN clients and VPN server. The VPN server relays data between VPN clients and internal application servers. Inside the LAN, communication between VPN server and application servers can also be protected using additional internal SSL tunnels, when high security required.

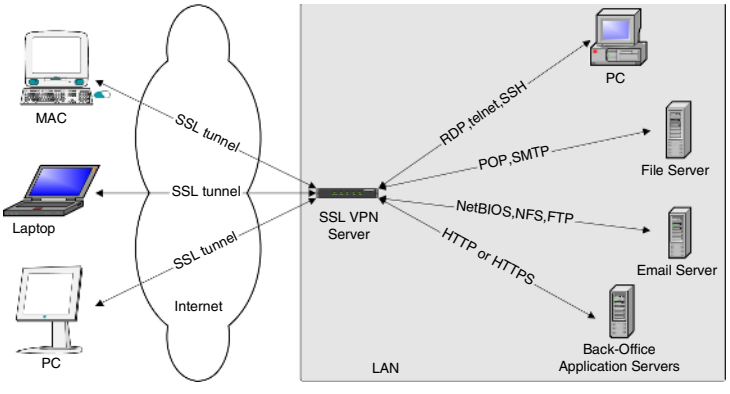


Fig. 2. VPN server is the common end of all SSL tunnels

All data flows of the VPN should be sent to VPN server, in which they are encapsulated or unwrapped server according to SSL protocol, and forwarded out. VPN server is computing intensive, and communication quality of the whole VPN is determined by computing power of VPN server. Research shows that the bandwidth utilizations of all open-source Linux-based SSL VPN solutions are less then 50 percent bandwidth of the 100Mb/s fast Ethernet, even using Pentium IV 2.0G platform [4,5]. Vendors usually have to install an expensive hardware SSL accelerator in VPN server to meet the demand of high-end applications or larger scale VPN cases.

For the purpose of removing the performance bottleneck of SSL VPN, an asymmetric SSL tunnel (AST) based VPN solution is proposed. In this solution, portion of

transfer overhead was distributed to disengaged internal application servers, and the overall VPN throughput can be improved greatly.

2 Asymmetric SSL Tunnel

Conventional SSL tunnel is symmetric: (1) both ends should encrypt outgoing date and decrypt incoming date; (2) data transferred in both directions is encapsulated using the same cipher spec. Figure 3(a) illustrates VPN server's work with symmetric SSL tunnels. VPN server authenticates client and creates SSL tunnel with it, and then relays the request from client to internal application server (See ①,②).After receiving response from application server, VPN server relays it to the client via SSL tunnel (See ③,④). If required, a SSL tunnel can be created between VPN server and application server to protect internal communication (See ⑤,⑥).

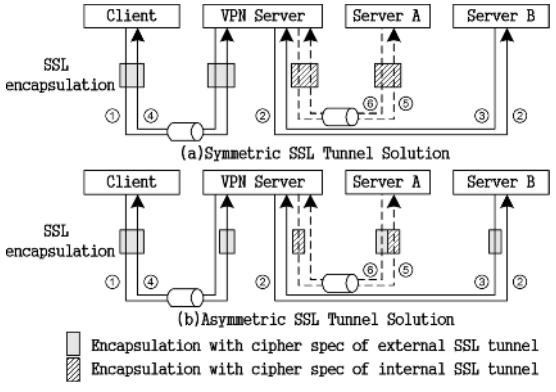


Fig. 3. VPN server works with symmetric SSL tunnels and asymmetric SSL tunnels

Because VPN server is the common end of all those symmetric tunnels, all data flows of VPN must pass through VPN server, and a lot of computational load is concentrated in VPN server. VPN server becomes the bottleneck, but CPU utilizations of internal application servers are always low. The more application servers exist, the worse. If some computation load can be transferred to those I/O intensive but CPU disengaged application servers, the overall performance of VPN should be improved.

VPN server must decrypt the request packets from clients, because it needs the plain request to do access controlling, virus scanning and finally forwarding it to application server. In contrast, the plaintext responses from application servers are not necessary for VPN server. According to that, a novel asymmetric SSL tunnel (AST) is proposed: If the cipher spec has been transfer to application server via secure tunnel, the application server can do SSL encapsulation immediately after the response IP packet generated (See ③,⑥ of Figure 3(b)), and VPN server just forwards the encapsulated packet to client.

From Figure 3(b), we can see that those asymmetric SSL tunnels have following features: (1) their ends can just only do data encryption or decryption; (2) data transferred in different directions can be encapsulated using different cipher spec.

Suppose SSL tunnels connecting VPN server and clients can be denoted by $\{T_1, T_2, \ldots, T_n\}$. $T_i = (If_i, Of_i)$, where If_i is input date flux of tunnel T_i (from client to application servers), Of_i is output data flux. The overall data flux transferred via internal SSL tunnels between VPN server and application servers is $\alpha\sum_{i=1}^{n} If_i + \beta\sum_{i=1}^{n} Of_i$, (0 α 1, 0 β 1). If no internal SSL tunnel used, $\alpha=\beta=0$; and if all data has to pass through internal tunnels, $\alpha=\beta=1$. Assuming that $\sum_{i=1}^{n} If_i : \sum_{i=1}^{n} Of_i = 1:k$

In traditional SSL VPN solutions, overall computational load of VPN server is

$$(\sigma+\lambda)\cdot\sum_{i=1}^{n}(If_i+Of_i)+\lambda\cdot(\alpha\sum_{i=1}^{n}If_i+\beta\sum_{i=1}^{n}Of_i)=[\sigma+\lambda+\lambda\alpha+k(\sigma+\lambda+\lambda\beta)]\cdot\sum_{i=1}^{n}If_i \quad . \tag{1}$$

Here λ is SSL encapsulation/decapsulation coefficient, and σ is relay coefficient. The relay coefficient σ is much less then λ ($\sigma << \lambda$). Assumed the processing power of VPN server is P, all time need for above computation is

$$t=\frac{[\sigma+\lambda+\lambda\alpha+k(\sigma+\lambda+\lambda\beta)]\cdot\sum_{i=1}^{n}If_i}{P} \quad . \tag{2}$$

So the theoretic maximum throughput of SSL VPN should be

$$Throughput=\frac{\sum_{i=1}^{n}(If_i+Of_i)}{t}=\frac{P\cdot(1+k)}{\sigma+\lambda+\lambda\alpha+k(\sigma+\lambda+\lambda\beta)} \quad . \tag{3}$$

When using asymmetric SSL tunnel, output data is SSL-encapsulated by internal application servers, and computation load of VPN server can be reduced as

$$\sigma\cdot\sum_{i=1}^{n}(If_i+Of_i)+\lambda\cdot\sum_{i=1}^{n}If_i+\lambda\cdot\alpha\sum_{i=1}^{n}If_i=[\sigma+\lambda+\lambda\alpha+k\sigma]\cdot\sum_{i=1}^{n}If_i \quad . \tag{4}$$

Assumed all application servers have enough free CPU resource to do SSL encapsulation, the overall throughput of VPN is still up to the power of VPN server, but it now increases to

$$Throughput'=\frac{P\cdot\sum_{i=1}^{n}(If_i+Of_i)}{\sigma\cdot\sum_{i=1}^{n}(If_i+Of_i)+(1+\alpha)\cdot\lambda\cdot\sum_{i=1}^{n}If_i}=\frac{P\cdot(1+k)}{\sigma+\lambda+\lambda\alpha+k\sigma} \quad , \tag{5}$$

and we have

$$\frac{Throughput'}{Throughput}=\frac{(\sigma+(1+\alpha)\cdot\lambda)\cdot\sum_{i=1}^{n}If_i+(\sigma+(1+\beta)\lambda)\cdot\sum_{i=1}^{n}Of_i}{(\sigma+(1+\alpha)\cdot\lambda)\cdot\sum_{i=1}^{n}If_i+\sigma\cdot\sum_{i=1}^{n}Of_i}=\frac{\sigma+\lambda+\lambda\alpha+k(\sigma+\lambda+\lambda\beta)}{\sigma+\lambda+\lambda\alpha+k\sigma} \quad . \tag{6}$$

For most of application servers, response data is usually much more than corresponding request ($If_i << Of_i$), that means $k >> 1$. We also has $\sigma << \lambda$, as mentioned above, so that computation load of VPN server will be reduced more then a half, and the overall throughput will be considerably improved.

3 Secret Management for AST Based VPN

In AST based VPN, internal application servers are connected to VPN server by symmetric SSL tunnel to receive secret messages. At VPN client logon, a master secret was negotiated, and VPN server assigned a new session ID for the connection. The secret and ID are cached in client and VPN server. When client connects internal application server, that session can be reused to avoid negotiation load.

The creation of new asymmetric SSL tunnel for client to access an internal server was illustrated in Figure 4 as phase 1. According to handshake protocol, VPN client simply specifies session ID of the session it wishes to reuse when sending the **Hello** message. VPN server checks in its cache to determine if it has state associated with this session. If the session state still exists in the cache, it uses the stored master secret and new random data of client and server to create a set of keys for new SSL channel. The client repeats the same process and generates an identical set of keys.

In SSL protocol, **change cipher spec** message causes the receiver to use new cipher spec to decrypt following data of the connection. Different from original protocol, that message was firstly generated by application server just after it receives new cipher spec from VPN server. VPN server relays that message to client. App server also generates the **finish** message using the new cipher spec. Upon client received those messages, it continues to act as SSL protocol described. After the finish message from client arrived at VPN server, the new AST was created.

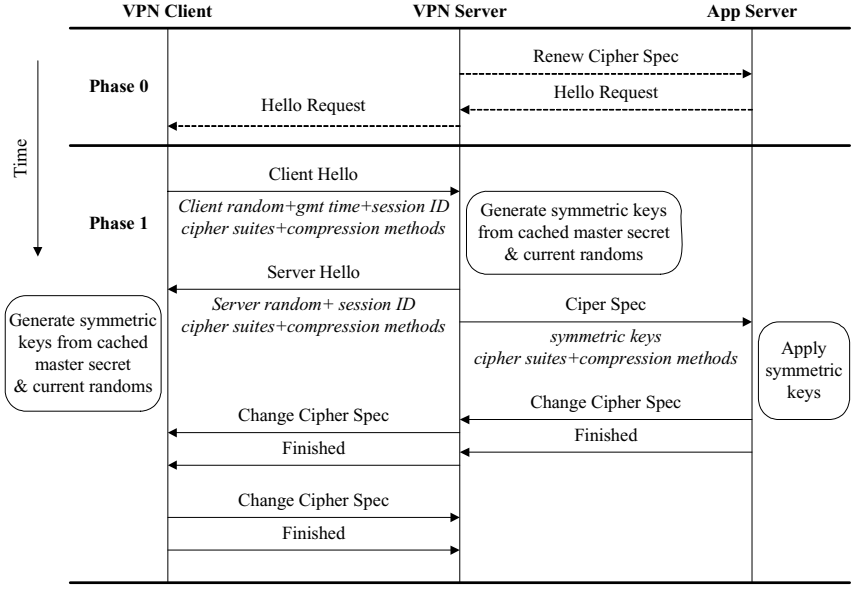


Fig. 4. Phase 1 is the process of creating a new asymmetric SSL tunnel, and phase 0 + phase 1 is the process of updating the parameters of a asymmetric SSL tunnel

In VPN application, ASTs always have long lifetime. For the purpose of enhance their security, it's suggested to update the symmetric keys periodically. There is a

hello request message from server defined by SSL protocol to initiate a renew process. It's profitable for VPN server to initiate a round robin renew process. In practice, SSL tunnels usually adopt cipher block chain (CBC) mode encryption algorithms, which are context depended, so that VPN server can't send that message directly. A simple renew scheme proposed for AST was depicted by the phase 0 of figure 4. When the time comes and VPN server is not busy, application server receives a renew cipher spec message from VPN server. Application server respond a hello request message, and that message was relayed to VPN client, and initiates a renew process, which is the same as session reusing process. After the renew process finished, cipher specs of all those three points are synchronized.

4 Kernel Optimization: IP Packet Engrafting and Fake TCP Header

SSL/TLS standards are defined over TCP protocol, so that SSL tunnel can also be regarded as secure TCP tunnel. In AST based VPN, upon receiving the SSL-encapsulated packet from application server, VPN server only needs to relay that packet to correct client via existing TCP tunnel. That's simple one-way relaying, and can be accelerated by IP packet engrafting algorithm.

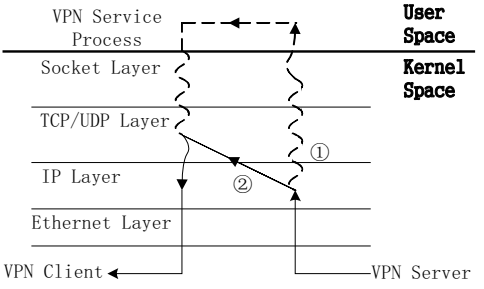


Fig. 5. IP packet Engrafting truncates the path of the SSL-encapsulated packet in VPN server

As figure 5 ① has shown, when a packet from application server arrived at the network interface card (NIC) of VPN server, it will be delivered up, passing through TCP/IP stack and socket layer one by one, and finally the VPN service process will receive it, which runs at user level. VPN service process chooses an outgoing tunnel according to the destination address of this packet, and then passes it down to NIC, layer by layer. Here exist at least two times of data copying between kernel space and user space, and also other processing of each layer, including generating new checksum in TCP protocol layer.

IP packet engrafting can cut down those load by shorten the processing path (see ② of figure 5). The SSL-encapsulated IP packet is directly insert to appropriate input buffer of TCP/IP stack upon receiving it, therefore the SSL-encapsulated packet turns to be a legal outgoing packet of TCP tunnel and will be sent out immediately. During that process, data copying between kernel space and user space is avoided [7,8].

In application server, a fake TCP header can be added into the outgoing packet, which carrying the SSL-encapsulated response packet to VPN server. That fake TCP header is filled with checksum of payload and other information to help IP packet engrafting in VPN server. Fake TCP header can help the IP layer of VPN server identify those encapsulated packets and simplify the processing in TCP layer, for example, eliminating memory allocation for TCP header and simplifying checksum computing.

IP packet engrafting reduces the relay coefficient from σ to σ′for outgoing packets, where σ′<σ. The throughput will increase as:

$$Throughput'' = \frac{P \cdot \sum_{i=1}^{n} (If_i + Of_i)}{\sigma \cdot \sum_{i=1}^{n} If_i + \sigma' \cdot \sum_{i=1}^{n} Of_i + (1+\alpha) \cdot \lambda \cdot \sum_{i=1}^{n} If_i} = \frac{P \cdot (1+k)}{\sigma + \lambda + \lambda\alpha + k\sigma'} \quad . \tag{7}$$

5 Implementation

OpenVPN[9] is a mature and full-featured open-source SSL VPN solution. In this solution, SSL tunnels are created to transfer IP packets over the Internet. Its program runs on almost all operation systems, such as Linux, Windows, FreeBSD, Mac OS X, and Solaris. Asymmetric SSL tunnel was implemented on the base of OpenVPN.

Originally, OpenVPN programs have two modes: VPN client mode and VPN server mode. Here we append an App server mode. Programs of the new mode will run on internal application servers. In our AST implementation, VPN server mode program runs on Linux platform, and a Linux kernel loadable module is written to do IP packet engrafting. The program uses a new system call to control this IP packet engrafting module, and its original SSL encapsulation capability has been reserved to relay those packets from application servers that haven't been encapsulated. Both client mode and app server mode programs run on any OS platform that original OpenVPN program does, so that a wide range of clients and application servers can be supported.

5.1 IP Packet Intercepting, Processing and Routing

Program of all those three modes uses TUN, a virtual device, to intercept IP packets, see figure 6. It is a system tunnel connecting TCP/IP stack and user processes. For user processes, it's a character device (for example ***/dev/tun0***), and for the network stacks of operation system, it's a virtual network interface (for example ***tun0***).

Every node of VPN, say VPN server, VPN client or application server, has a virtual IP address that is valid within VPN, so call it *VIP*. In VPN client, outgoing IP packets, whose destination addresses are *VIP*s of application servers, are automatically routed to virtual network interface of TUN. Client mode OpenVPN daemon receives those packets from the character device, then encapsulates and sends them to VPN server via the SSL tunnels. Incoming packets from the SSL tunnels go through the contrary path, and their payloads, the unwrapped packets, are finally accepted by the TCP/IP stack as if they came from the real physical NIC.

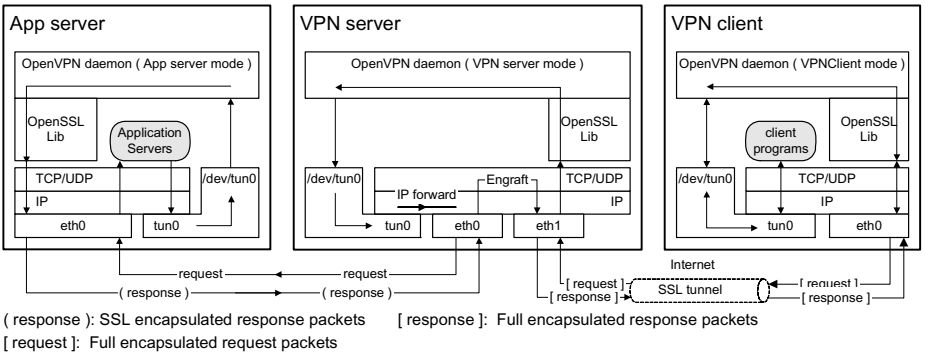


Fig. 6. TUN device is system tunnel between processes and TCP/IP stacks

VPN server unwraps the request packets from client, and relays them to application servers. Application server generates response packets using *VIP* of the client as their destination addresses. In original OpenVPN solution, VPN server works as an ARP proxy or a gateway with the IP forwarding capability enabled, so that those response packets are routed to VPN server by data-link protocol, and then forwarded to the TUN device of VPN server. OpenVPN daemon encapsulates them and sends them to client via conventional symmetric SSL tunnels.

When AST was used, application server and VPN server work at a different way. OpenVPN program of app server mode runs on application server as a daemon. In initialization phase, it automatically searches VPN server, and then creates a SSL tunnel with the VPN server to receive cipher spec and other information for local SSL encapsulation use, as has mentioned in chapter 3. That information is stored in an asymmetric SSL tunnel information table (ASTIT) indexed by the client *VIP*s, see table 1. The SSL tunnel is also used to transfer client requests to application server if internal security required. App server mode daemon intercepts response packets via TUN device and encrypts them using corresponding cipher specs according to SSL Record protocol. After that, the daemon wraps those SSL-encapsulated packets into TCP packets and sends them to VPN server. VPN server engrafts them into appropriate AST connections and relays them to VPN clients.

Table 1. Record format of AST Information Table and Engrafting Infromation Table

Table Name	Record Format
ASTIT	Client VIP : AST Port : Session ID : Cipher Suit : Symmetric Keys
EIT	Client VIP : App Server VIP : Local Port : Remote Port : Client Public IP

5.2 IP Packet Engrafting

If outgoing packets of application server are to be engrafted into ASTs, they are written to raw socket and sent to VPN server. App server mode daemon prepares the IP headers and fake TCP headers of those packets by itself.

The destination address of those TCP packets should not be *VIP*s of clients, otherwise they would be intercepted by the TUN device of application server and encapsulated again and again, never have the chance to be sent to VPN server. But VPN server needs destination information to choose correct tunnels for those packets. So that the header of those outgoing packets can be set as:

```
Output.DstIP = RIPLG;  Output.DstPort = EngraftPort.
Output.SrcIP = VIPC;   Output.SrcPort = ASTPort.
```

Where *RIPLG* is the real IP address of VPN server in LAN, and *EngraftPort* is the port listened by IP packet engrafting module in VPN server. *VIPC* is the *VIP* of destined client, and *ASTPort* is VPN server end port of the AST to be engrafted on.

OpenVPN daemon of VPN server mode maintains a engraft information table (EIT) in the IP packet engrafting module. That table records necessary information for engrafting input packets into correct AST, see table 1. In that table, *local port* and *remote port* are the port numbers used by the tunnel.

Once IP packet engrafting module received a input packet from the *EngraftingPort*, the tunnel information table is queried to find a asymmetric SSL tunnel, whose *local port* = *Input.SrcPort*. The packet is inserted to the input buffer of TCP protocol layer after following modifications:

```
Output.DstIP = Client Public IP ;
Output.SrcIP = Public IP shared by LAN ;
Output.DstPort = Remote Port ;
Output.SrcPort = Local Port ;
Output.Checksum = Input.Checksum + ChecksumPatch ;
Output.SeqNum = TCPConn[Local Port].SeqNum ;
Output.Ack = TCPConn[Local Port].Ack ;
Output.Win = TCPConn[Local Port].Win .
```

Here *ChecksumPatch* revises the TCP checksum. Because only the header has been modified, the *ChecksumPatch* reflects the difference between new header and the old one. The *TCPConn[Local Port]* is a kernel structure (TCP control block), it represents the TCP connection of the AST, whose local port is *Local Port*. That structure contains some parameters of the connection: next sequence number (SeqNum), acknowledge number (Ack), window size (win) and so on. Packet engrafting will results in an increment of the sequence number:

```
TCPConn[Local Port].SeqNum =
 (TCPConn[Local Port].SeqNum + Input.Length) mod 232.
```

6 Performance Evaluation

Performance evaluation is focused on testing the improvement of SSL VPN throughput and latency brought by asymmetric SSL tunnel. We only evaluate the performance of VPN after clients have already logon, and SSL tunnels have been created for clients to access internal application servers. The cipher specs of those SSL tunnels never changed after creation. To evaluate the most common case, all SSL tunnels use lzo as the compressing method, and DES-CBC-SHA as the cipher and MAC algorithm.

In our experiments, a LAN and 4 VPN clients are used to construct a SSL VPN. There are a VPN server, a gateway and 4 ftp servers in the LAN, connected via a 100Mbps switch. The gateway and 4 VPN clients are also connected via another 100Mbps switch. Table 2 lists the hardware and software configurations.

Table 2. Hardware and software configuration

Name	Qty.	CPU	Mem	OS	Software
VPN server	1	Intel PIII 800	256MB	Linux 2.4	openvpn(server),mod_IPengraft
LAN gateway	1	AMD 1.7G	256MB	FreeBSD 5.0	netd,ipfirewall
FTP servers	4	Intel PIII 800	256MB	Linux 2.4	openvpn(app srv),vsftpd
VPN clients	4	AMD 1.7G	256MB	Linux 2.4	Openvpn(client),dkftpbench

6.1 Throughput Evaluation

FTP service is the common application of VPN, and FTP protocol has a simple commend set, so that we choose FTP application to test the throughput of SSL VPN. Dkftpbench[10] is an open-source FTP benchmark program inspired by SPECweb99. It simulates many simultaneous ftp clients, which repeatedly downloads the same file

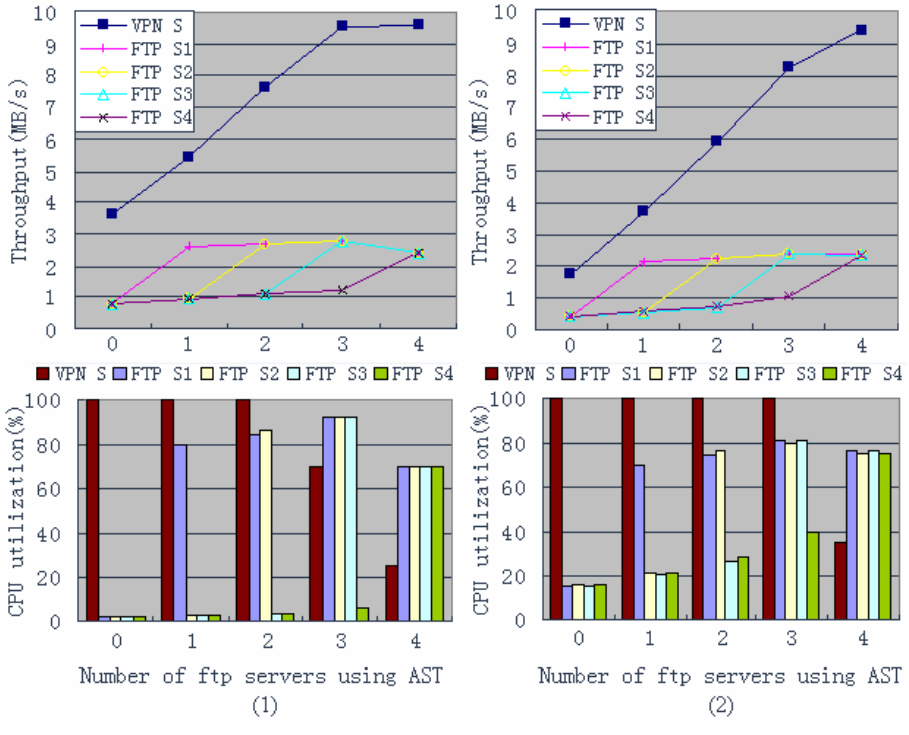


Fig. 7. Throughput test results of two security levels: (1) normal, (2) internal security required

from the evaluated ftp server. At the end of each cycle, the client's average downloading speed in that cycle is reported. In our tests, dkftpbench is modified to report the overall downloading speed periodically. In each VPN client, the dkftpbench simulates 50 FTP clients to download a 5M bytes file from a corresponding ftp server, and all ftp clients download files with their best effort.

To eliminate influence from file system and hard disk, all the ftp servers have been warmed up, and the requested files have been loaded into the memory before test. During the test, make sure that the throughput of VPN is drained off, and CPU utilizations of all VPN clients are lower than 100%.

In the beginning of test, no AST enabled, and conventional symmetrical SSL tunnels are used to transfer responses of ftp servers to VPN clients. When all these parameters turn to be stable, VPN throughput, CPU utilizations of VPN server and ftp servers, we record their values. After that, app server mode OpenVPN daemons start work one by one in ftp servers, and stable value of above parameters also be recorded for comparison.

Graph (1) of figure 7 illustrates test results in the case that internal security of LAN was not required. When there is no AST used, CPU utilization of VPN server is 100%, but CPU utilizations of internal ftp servers are very low, and the overall VPN throughput is just 3.64MB/s. After one ftp server has OpenVPN daemon started, its CPU utilization increased, and because its response packets ware relayed directly at the TCP/IP stack of VPN server by IP engrafting module, throughput of that ftp server also increased considerably. The CPU resource saved by asymmetric tunnel also contributed to higher throughput of other ftp servers. From the graph, we can see that the throughput and CPU utilization of other ftp servers also increased a little.

When more internal ftp servers use ASTs to transfer response data, overall throughput of VPN increased. After 3 internal servers start using ASTs, the overall VPN throughput achieves 9.64MB/s, which is almost the limitation for SSL tunnel over 100M network, and VPN server started to have surplus CPU resource. When all the 4 internal servers used asymmetric SSL tunnels, VPN server had more surplus CPU resource, and CPU utilization of other internal servers also decreased a little, but the overall VPN throughput not varied.

The throughput improvement brought by AST is more remarkable in the case that internal security of LAN is required. Graph (2) of figure 7 depicts test results in that case. Because asymmetric SSL tunnel solution virtually eliminated one times of SSL encapsulation and decapsulation for every response packet. The overall throughput of SSL VPN increased from 1.76MB/s to the limit of 100M networks.

6.2 Latency Evaluation

We use *ping* program at a VPN client to measure response time of internal server. Internal server was pinged 50 times for each test. The mean value of round trip time was taken. Difference between the mean and median is less than one standard deviation.

If no internal security required, latencies of both conventional tunnel and asymmetric tunnel are the same: 5.1ms. When internal security required, latencies of both tunnels increased. AST's latency is 7.7ms, and symmetric tunnel's latency is 11.2ms. In this case, AST solution reduced the latency by about 31%. Using asymmetric SSL tunnel, one times of SSL encapsulation and decapsulation processing are eliminated for every response packet, and that caused the latency reduction.

7 Conclusion

Asymmetric SSL tunnel based VPN has a significantly higher overall throughput than traditional SSL VPN. If no internal security required, the improvement comes from shifting portion of transfer loads from VPN server to internal servers. If internal security required, portion of computation load of VPN server can be eliminated by using appropriate cipher spec for outgoing packets in internal server. In both cases, transfer overheads of outgoing data are removed from VPN server. The more proportion the outgoing data has, the more improvement achieves.

The security of AST based VPN doesn't degenerated. Its secrete management scheme can synchronize cipher specs of multipoint, and higher security requirement is supported. In addition, if no internal security required, asymmetric SSL tunnel solution causes the outgoing packets of application servers to be encrypted, and the security in LAN was improved.

AST based VPN solution doesn't need any modification on internal server programs. In our implementation, the app server mode OpenVPN daemon runs on internal server and automatically searches VPN server. No additional configuration is required. Both client mode and app server mode programs can run on many OS platforms, such as Linux, Windows, FreeBSD, Mac OS X, and etc., so that a wide range of clients and application servers can be supported.

References

1. Gartner Company. http://www3.gartner.com
2. Alan O. Freier, Philip Karlton: The SSL Protocol Version 3.0 [EB/OL]. http://wp.netscape.com /eng/ssl3/draft302.txt, (2004)
3. T. Dierks and C. Allen, RFC2246: The TLS Protocol Version 1.0. http://www.ietf.org/rfc/rfc2246.txt, Jan. 1999.
4. S. Khanvilkar and A. Khokhar: Virtual private networks: an overview with performance evaluation. Communications Magazine, IEEE Vol. 42, Issue 10, (2004) 146 – 154
5. S. Khanvilkar and A. Khokhar: Experimental evaluations of Open-Source Linux-based VPN solutions. ICCCN'04, (2004)
6. Apostolopoulos, G., Peris, V., Saha, D.: Transport layer security: how much does it really cost? INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 2, (1999) 717 - 725
7. Di Santo, M., Ranaldo, N., Zimeo, E.: Kernel implementations of locality-aware dispatching techniques for Web server clusters. Proceedings of IEEE International Conference on Cluster Computing (CLUSTER'03). (2003)154 – 162
8. Kobayashi, M., Murase, T.: Asymmetric TCP splicing for content-based switches. Proceedings of IEEE International Conference on Communications (ICC 2002). Vol.2 (2002)1321-1326
9. OpenVPN: http://www.openvpn.net
10. dkftpbench: http://www.kegel.com/dkftpbench/

Multihomed Routing in Multicast Service Overlay Network

Xiao Chen and Weinong Wang

Regional Network Center of East China,
Department of Computer Science and Technology,
Shanghai Jiao Tong University, China
{shawn, wnwang}@sjtu.edu.cn

Abstract. Most existing overlay multicast approaches assume a singlehomed model, where each overlay node has a unique access link in the physical network. This assumption is usually not applicable to the routing problem in multicast service overlay network, which may comprise multihomed multicast service nodes(MSN). We address this issue by proposing not only a multihomed proxy model but also a two-phase solution to the routing problem under this model. First we choose a proper physical path for each overlay link based on the current network status and the routing policy. Then the routing process calculates the multicast tree according to the previous selection. Through simulations, we prove that our approach makes the overlay multicast routing more efficient in a multihomed environment.

1 Introduction

Overlay multicast means that the multicast is carried out in an overlay network, which is composed of a subset of underlying physical nodes. Each link in an overlay network amounts to a unicast path between the connected two nodes. Because each pair of nodes have a unicast path between them, the overlay network is in a full-meshed form. Each node in an overlay network is able to forwarding packets at the application layer so that it can enable multicasting without changing the underlying network. The data for multicasting is only replicated at branching points in an overlay multicast tree and sent from one node to another through an ordinary unicast transmission. In this paper we assume that all these multicast service nodes (MSNs) are stationary nodes deployed in a multicast service overlay network. In contrast with an end-host based overlay, a service overlay network provides a robust multicast service while still keeping a high flexibility in routing.

Since the number of MSNs is much smaller than that of ordinary routers, all MSNs should be located at critical points in the physical network so that they can form a cost-effective overlay network. Therefore it's most likely that a lot of MSNs will be multihomed nodes. In other words, these MSNs will have multiple access links to the Internet. The overlay service provider has good reasons for such a deployment. First of all, multiple access links reduce the average distance

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 125–136, 2006.
© Springer-Verlag Berlin Heidelberg 2006

from one MSN to all other peers. This is inferred from the observation that the shortest paths from a multihomed node to others usually go through different access links in the network. The second reason is that if two MSNs belong to two adjacent domains respectively, their communication can be seriously affected by the inter-domain link bottleneck. Based on this consideration, the overlay service provider is encouraged to locate the MSNs in between different domains so that they can have multiple access links to each of them. Such a deployment can guarantee a high capacity over the overlay links. In addition to above benefits, multiple access links also promise a high availability for the MSNs. Since all multicast traffic depends on these MSNs, any breakdown may cause serious performance degradation. The installation of multiple access links is a better choice for keeping an MSN always online.

In spite of these facts, the overlay multicast routing problem is usually considered under a singlehomed proxy model in most literatures[1][6][5]. Unlike IP multicast[4], overlay multicast usually has a link stress[2] larger than one. In this model, an MSN's capacity in data forwarding is rendered as its node degree bound. Two MSNs can be connected in a multicast tree by an overlay link as long as both of them have residual degrees, while the overlay link equals to the shortest unicast path between them. This model is good if each MSN only has a unique access link to the Internet. In this case one MSN can speak to another only if its unique access link has enough capacity. When an MSN has multiple access links, this model is no longer suitable. We illustrate an example below in Fig. 1 for demonstration.

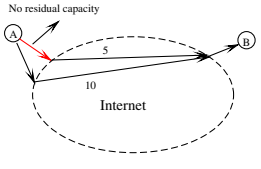


Fig. 1. Multiple access links to the Internet

In this figure, MSN A has two access links to the Internet. One link, which stands in the shortest path from MSN A to MSN B, has exhausted its access bandwidth. The other link, however, has plenty of residual capacity. We assume that MSN B also has enough access bandwidth for connection. Then these two MSNs can be connected in the tree but not by the shortest path. Since the routing algorithms[1][6] assume a fixed distance for the each overlay link, they are not applicable to such an instance.

The objective of our work is to propose not only a multihomed proxy model to describe the overlay multicast routing problem accurately but also a systematic approach to solve this problem under our model. Briefly speaking, we break the whole solution into two coupled processes. First we choose a proper physical path for each overlay link based on the current network status and the routing policy. Then the routing process calculates the multicast tree according to the

output of the first process. In our model the capacity of each MSN is translated to the access link's capacity rather than the node's degree bound so that our routing algorithm can achieve a high utilization among the multiple access links.

The rest of this paper is organized as follows. Section 2 proposes the multihomed proxy model and draws a comparison between our model and the original one. In Sect.3 we enumerate three different path selection algorithms and analyze their performance from different perspectives. We illustrate the combination of path selection and multicast routing by a modified CT(compact tree) algorithm in Sect.4. Section 5 shows our simulation results, while the conclusions are drawn in Sect.6.

2 Overlay Network Model

2.1 Singlehomed Proxy Model

The overlay network is usually considered as a complete graph, where each edge stands for the shortest unicast path between the corresponding two nodes. Thus the resource limitation for the access bandwidth could be directly transformed to the degree constraint for each node in the generated multicast tree. This model is perfect under some environment, where each overlay node has only one unique access link to the Internet, such as end-host overlay multicast. However this model is not suitable when any of the MSNs has multiple access links to the Internet. The notion could be illustrated in the following figures.

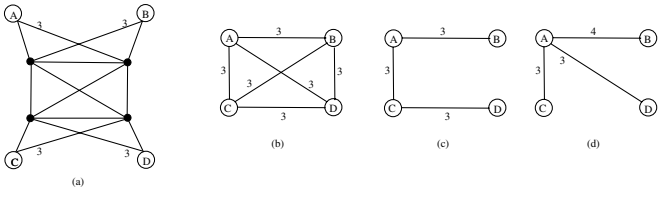


Fig. 2. Topologies for overlay and physical network

In Fig. 2(a) nodes A to D are four MSNs in the physical network, each of which has two access links to the Internet. Each unlabelled link has a uniform cost of one. According to the fully meshed model, the overlay network among nodes A to D should be depicted as Fig. 2(b). Note that in this figure only one access link of each MSN is used to construct all overlay links in the overlay network. The longer access links of cost 3 are totally neglected from the full-meshed overlay network model. Suppose in each session the upper bound of the link stress for each access link is 2. In other words, we assume that no access link could carry more than 2 duplicate data copies in one multicast session. Based on the model in Fig. 2(b) one possible solution for the overlay multicast tree is shown in Fig. 2(c), which spans all MSNs and has a diameter of 9. If we take the longer access

link into consideration, the result will change. In Fig. 2(d) we adopt the longer link in our tree construction and obtain a multicast tree with a smaller diameter of 7 while not violating the link stress limitation. In this case we can see that the singlehomed model cannot help us to get the best result in the overlay multicast routing problem.

2.2 Multihomed Proxy Model

While multiple access links enhance the availability and utilization of the MSNs, they introduce several new problems. As we have seen in the above example, since the access bandwidth for each MSN in each multicast session is allocated on the specific access links, the degree constraint model for the access resource limitation in [3] is not applicable any more. The access bandwidth consumption is not represented by the degree of the node in the multicast tree but by the actual link load. Therefore the resource allocation balancing is performed on a link basis which makes a big difference in the generated tree topology. In addition, since two MSNs can be connected through different pairs of access links, the network distance between the two nodes could vary due to different choices. In order to make a tradeoff between the network latency and the resource allocation balancing, it is important to choose the proper unicast path as the overlay link for multicast tree construction. This effort also adds to the overhead of the routing process.

In order to solve all above problems, we first need another model for the overlay network. Suppose the overlay network in our research is defined as a ternary structure $OG = (V, A, E)$. V represents the set of MSNs while A stands for the set of access links attached to the MSNs. Let ve denote the function that maps a certain access link to its attached MSN such that for every $l \in A$ there is one and only one MSN $v \in V$ and $ve(l) = v$. We also use $le(v)$ to denote the set of access links of MSN $v \in V$. E is a set of overlay links, each of which corresponds to a shortest unicast path between two MSNs through a pair of access links $(l1, l2)$. Note that in our model there may exist multiple overlay links between two nodes due to different choices of the access links. Function $c(l1, l2)$ calculates the cumulative cost along the shortest unicast path through access links of $l1$ and $l2$ between the nodes $ve(l1)$ and $ve(l2)$. For each access link l, we use $s_{max}(l)$ and $s_T(l)$ to represent the maximal available bandwidth resource and the consumed bandwidth for tree T over link l respectively. To simplify our model, when two nodes is connected by one link directly, this shared access link is counted twice in A. Based on these definitions, we will introduce a revised version of the MDDL and LDRB[1] [1] problems as formulations of our specific overlay multicast routing problem.

Definition 1. *Minimum diameter, link stress limited spanning tree (MDLSL): Given an overlay network model $OG = (V, A, E)$, a link capacity bound $s_{max}(l) \in$*

[1] These two problems are minimum diameter, degree-limited spanning tree problem and limited diameter, residual-balanced spanning tree problem respectively.

N for each access link $l \in A$ *and a cost* $c(l1, l2)$ *for each overlay link* $(l1, l2) \in E$*, find a spanning tree T of OG of minimum diameter, subject to the constraint that* $s_T(l) \leq s_{max}(l)$ *for all* $l \in A$

Suppose that each node in V only has one attached access link in A. Then the MDLSL problem will amount to the MDDL problem which has been proven to be NP-complete[3] for a fixed degree bound. Consequently our MDLSL problem is also NP-complete under an assumption of a fixed link stress bound.

The second formulation of the problem aims to find the most balanced allocation of the link stress over all related access links. Similar to the definition of residual degree in [1], we define the residual capacity at access link l with respect to a tree T to be $rest(l) = s_{max}(l) - s_T(l)$. Then the most balanced allocation indicates a maximal smallest residual capacity among all access links. Such an allocation promises a least likelihood of the session block.

Definition 2. *Limited Diameter, residual capacity balanced spanning tree problem (LDRCB):*
Given an overlay network model $OG = (V, A, E)$*, a link capacity bound* $s_{max}(l) \in N$ *for each access link* $l \in A$ *, a cost* $c(l1, l2)$ *for each overlay link* $(l1, l2) \in E$ *and a diameter bound of B, finding a spanning tree T of OG with diameter* $\leq B$ *that maximizes* $min_{l \in A} rest(l)$*, subject to the constraint that* $s_T(l) \leq s_{max}(l)$ *for all* $l \in A$

LDRCB is also an NP-complete problem because its special case is the LDRB problem, which has been proven NP-complete [1].

We try to imitate the description of MDDL and LDRB when defining our problems in order to show the differences and association between the two versions of problem definitions. One thing they have in common is that they all try to find a tree spanning all overlay nodes. However our version performs the cost calculation and resource allocation in respect of access links rather than overlay nodes.

3 Path Selection Algorithm

In a lot of overlay network architectures, the overlay link between two nodes corresponds to the shortest path between them. This assumption works well when the network latency is considered as the primary parameter. However this is inadequate when several other factors are taken into consideration, such as balanced bandwidth consumption and bandwidth demands for each session. As the bandwidth requirement may vary in different multicast session, the shortest path may not be the best choice. Even though it has enough capacity, it should be avoided if the shortest one has been overused while other paths of a similar length are left unused. Therefore a proper path selection algorithm is important before the routing algorithm could do anything. We will introduce three path selection algorithms as follows for the MDLSL and LDRCB problems.

3.1 Shortest Path First

We assume that the partial network between any two access links has sufficient capacity for any session request so that the main bottleneck remains on all access links. This assumption is usually true when the MSNs are located near the backbone. Thus an overlay link $(l1, l2) \in E$ amounts to the shortest underlying path between $ve(l1)$ and $ve(l2)$ through $l1$ and $l2$. Let $cap(l1, l2)$ denote the minimum of $s_{max}(l1)$ and $s_{max}(l2)$ so that it represents the maximum bandwidth requirement the overlay link $(l1, l2)$ can satisfy. Given a session request with a bandwidth requirement of b, we call an overlay link $(l1, l2)$ an eligible one for this session if and only if $cap(l1, l2)$ is greater than b.

Intuitively the shortest possible path algorithm is to select the shortest eligible path as the overlay link for the concerned two nodes without considering whether the residual capacity for each access link is balanced. One straightforward way is to remove all ineligible access links from the underlying network, which has no enough bandwidth for the request, and calculate the shortest path between these two nodes. The corresponding pair of access links for this path will stand for the overlay link for the two nodes in our model. This approach will surely try to assign a minimum cost for each overlay link. It is intended as a greedy heuristic for the MDLSL problem, which tries to find the minimum diameter tree as long as there is enough resource.

3.2 Broadest Path in k-Shortest Ones

In most cases, a longer tree diameter doesn't matter too much as long as it is within a specific bound. Based on this notion we give a heuristic here for the LDRCB problem, which tries to optimize the workload allocation as well as the tree diameter. Suppose nodes $v1$ and $v2$ have $n1$ and $n2$ eligible access links for a session request respectively. Then there are potentially $n1 * n2$ candidate paths for the overlay link between these two nodes. Sort all these eligible paths in ascending order according to their distance. The path with the largest capacity in the first k paths then will be our choice for the overlay link.

The value of k determines the preference of this algorithm between workload balancing and diameter minimization. If k equals to one, then it amounts to the shortest path first algorithm. If we assign $n1 * n2$ to k, then this approach is to find the path with the largest capacity. According to our empirical results, a value of four or five will suffice in respect of both requirements.

3.3 Alternative Path Algorithm

Both of the above two algorithms select only one underlying path as the overlay link. This could be insufficient in some cases. For example, one overlay link may run out of bandwidth before it is used for tree construction because it shares the same access link with some other overlay link, which has already consumed the access bandwidth in the tree. Our solution to this issue is to have two underlying paths prepared for each overlay link. The primary path has a larger capacity

while the secondary one has a shorter distance. Therefore the upper layer overlay routing algorithm may have an alternative when some parameter of a certain overlay link is unsatisfactory. Although this approach cannot avoid all those situations stated above completely, it can drastically reduce their occurrence according to our simulation results.

The detailed algorithm is as follows. We choose the broadest path in the k shortest ones to be the primary path. Unless the primary path is also the shortest path, another shorter one will be selected as the secondary path. Then in most cases the primary path has an advantage in terms of capacity, while the secondary path has a shorter distance. If the primary path is also the shortest one, the secondary path should be the shortest one among those, each of which shares no access links with the primary path. This is because if the primary path is already the shortest one, we need the secondary path only if the primary one is running out of resource. Therefore our algorithm makes it much more likely that the secondary path still has enough capacity when the primary one is already congested.

3.4 Comparisons

The first aspect we want to compare between the above algorithms is their time complexity. For convenience we refer to the above three algorithms as S algorithm, B algorithm and A algorithm respectively. We assume that all MSNs are stationary nodes in the physical network and each link has a fixed value of network delay. Since the overall distance for each unicast path between a given pair of MSNs can be calculated prior to the execution of the path selection algorithm, the only operation for these algorithms is to find the proper paths for all overlay links. Suppose each MSN has at most m access links to the Internet. Then the number of underlying paths between two MSNs is at most m^2. For the S algorithm, it is only necessary to check all m^2 paths and pick the shortest one from them. So its time complexity is $O(m^2)$. In order to find the broadest path in k shortest ones, we not only have to sort all m^2 paths by their delay but also need to go through the k shortest ones to pick out the broadest path. Its total time complexity should be $O(2m^2 logm + k)$. Because the A algorithm tries to find a secondary path, it will need additional $O(klogk)$ steps to previous operations to sort the k shortest paths by their distance. Thus its time complexity will be $O(2m^2 logm + klogk + k)$. In practice the values of m and k are very small even in large networks so that the most complicated algorithm is about three times as complex as the simplest one.

In addition to the time complexity, we also need to know how close the selected path is to the shortest possible path in different algorithms. For simplicity we assume here that the capacity of a path has nothing to do with its distance in the latter two algorithms. It is obvious that the S algorithm will always give the shortest possible path. As for the B algorithm, if we sort the k shortest paths in an ascending order by their distances, a random variable X can be used to represent the order of the broadest path in this list. According to our assumption, $P(X = i) = 1/k(1 \leq i \leq k)$. Therefore the expected value, $E(X)$,

will be $(1+k)/2$. It means that on average the output path will be the $(1+k)/2\ th$ shortest path among all k paths. If we define a similar random variable Y for the secondary path in the A algorithm, its expected value, $E(Y)$, is approximately $k/4$, which is closer to the shortest path.

The third criterion for a path selection algorithm is how much it could help the overlay multicast routing algorithm to reduce its rejection rate. Because a session request could be rejected either by a stringent diameter constraint or due to the resource exhaustion over some overlay links, a reasonable selection of the underlying paths should make the overlay network satisfy as many requests as possible. Although this criterion is even more important than the previous two, it is hard to measure without regard to the upper routing algorithm. We will combine all three algorithms with a certain routing algorithm introduced in section IV and compare their simulation results in section V. Intuitive the S is suitable for a strict diameter limitation while the B algorithm accepts more session requests under a loose diameter constraint. Since the A algorithm offers a secondary choice when the primary one is not satisfactory, we expect that its performance is better than the other two in most cases.

4 Routing Algorithm

4.1 Integrated Routing Algorithm

The path selection algorithm and the multicast routing algorithm work in two different layers. The former selects one path among multiple paths between two MSNs to be the overlay link. The latter selects a collection of overlay links from the full-meshed overlay network to build the tree. While our research here focuses on path selection algorithms, their performance can't be evaluated without regard to the upper layer routing algorithm. A good path selection algorithm should help the routing algorithm to reduce the session rejection rate while keeping a balanced utilization among different paths. A bad one may lead to network congestion at some points but a low resource utilization somewhere else. We can't conduct such measurements unless we bind the path selection algorithms to a certain multicast routing algorithm. For simplicity we choose the CT(compact tree) heuristic introduced in[1] for this purpose.

The CT algorithm is similar to Prim's minimum spanning tree. It first chooses a node as the root and proceeds to add a node to the tree, which leads to a minimum diameter. In detail, it uses $d(v)$ to denote the distance of the longest overlay path from node $v \notin T$ to any other node $u \in T$. The node v with the minimum value of $d(v)$ is chosen each time to be connected to the tree T through node $n(v)$ if the degree constraint is not violated. This process goes on until all nodes are connected to the tree.

Because the original CT only considers the degree constraint, we need to transform it to recognize link level bandwidth limitation. We order that the path selection algorithm supplies an m-element capacity vector Cap for an n-node overlay network ($m = |A|$). This vector corresponds to the capacity of all access links used in an overlay network. $Link(v, u)$ denotes the index of the access link from

node v to u on the overlay link (v, u). Then each time an overlay link (v, u) is used to build the tree, the value of $Cap[Link(v, u)]$ and $Cap[Link(u, v)]$ is updated to $Cap[Link(v, u)] - req$ and $Cap[Link(u, v)] - req$ respectively where req stands for the bandwidth requirement of the session. By this way an overlay link (v, u) could be used if and only if its related two access links have enough capacity.

Therefore a basic path selection algorithm should at least supplies the value of $dist(v, u)$ and $Link(v, u)$ (also $Link(u, v)$) for each overlay link (v, u) as well as a capacity vector Cap for all involved access links. All these information informs the overlay network layer of the distance of each overlay link, the access links of each overlay link and the capacity of each access link. According to these knowledge a routing algorithm can build a multicast tree without knowing the details of the underlying network. Since the alternative path algorithm calculates a secondary path for each overlay link, it needs another two functions of $dist'(v, u)$ and $Link'(v, u)$ to denote the distance and access links of the secondary path. We will illustrate a modified CT algorithm in Fig. 3 to show how a secondary path can be used in tree building.

In the above algorithm, some secondary paths are used to substitute their primary paths for a shorter tree diameter when the diameter limitation is violated during the tree building process. We call it a shrinking process. In the shrinking process, the value of $d(u)$ for some $u \in W$ may be changed. In order to reduce the time complexity, we can record a list of nodes on each overlay link such that only those nodes need to update their longest path distance when this overlay link is shrunk. In addition each step in above algorithm is executed only if the overlay link has enough capacity in vector Cap.

The routing algorithms combined with our first and second path selection algorithm are very similar to this one except that they don't have a shrinking process. Therefore we have shown a general example for integrating a path selection algorithm with an overlay multicast routing algorithm.

5 Simulations

In this section we will evaluate the performance of different path selection algorithms through several simulations. As we mentioned above, their practical performance can't be tested without a certain routing algorithm. Thus we will use the CT routing algorithm introduced in Sect.4 for this purpose.

5.1 Performance on Rejection Rate

A session is rejected if we can't find a multicast tree in the overlay network to satisfy the tree diameter limitation. This could happen for two reasons. One is that the tree diameter limitation is too strict for the session members. Another reason is that the existing sessions have exhausted the bandwidth of most bandwidth of these MSNs so that it is hard to build a tree with short paths. A low rejection rate is significant because it proves that the algorithm is viable in practice. According to above considerations we design two experiments to evaluate

```
Input:
  An overlay graph OG=(V,A,E)
  Values of dist,dist',Link,Link'
  Capacity Vector Cap[1..m] m=|A|
  Diameter Limitation B
  Bandwidth Requirement req
Output:
  Tree with the minimum diameter
  {No element in vector Cap should be negative
  after an overlay link is inserted in the tree}
Program Tree_Building (root u)
   Let Dist(v1,v2) = dist(v1,v2) for all node pairs in V
   Let d(v) = Dist(v,u) for all nodes v in V
   Let n(v) = u all nodes v in V
   Let T = {W = {u}, L = {}}
   While W != V do
     Let v be the node with smallest d(v) in V - W
     If d(v) > B Then
       Shrink the longest overlay path from v to any node in W
       Recalculate the value of Dist, Cap and d(v) involved in the
       shrinking process
     Include v in W and include (v, n(v)) in L
     If use primary path Then
       Cap[Link(v,n(v))] -= req
       Cap[Link(n(v),v)] -= req
     Else
       Cap[Link'(v,n(v))] -= req
       Cap[Link'(n(v),v)] -= req
     Upadate d(v') for all other v' in V - {v}
```

Fig. 3. CT algorithm modified for alternative paths

the performance of different path selection algorithms on rejection rate from different perspectives. In the first experiment, we define the diameter bound to be the proportion between the length of the longest overlay link among the session members and the tree diameter. Each session has 10 members. The k value for the B algorithm is 4 in all our experiments. Based on these simulation settings, the rejection rate of the first 100 session requests is shown in Fig. 4(a).

From this figure we can see that the S algorithm always has a least rejection rate. This is because 100 sessions is not a big burden for our designed overlay network. When there is enough capacity, the S algorithm is surely the first choice especially for a strict diameter bound. We also find that the curve for the A algorithm is very close to that of the S algorithm. One more thing to mention is that the curves drop sharply when the diameter bound is above 5. This indicates that the relative delay penalty is less than 3 in our overlay multicast.

The second experiment draws the curves as the session number increases. The diameter bound we use here is 6. The result is shown in Fig. 4(b). According

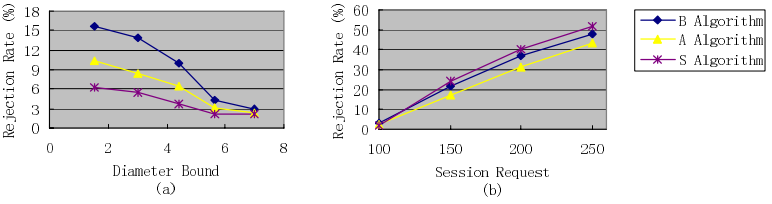


Fig. 4. (a)Rejection VS diameter bound (b)Rejection VS session requests

to this result the performance of the S algorithm is no longer so satisfactory as it is in the previous simulation. When an increasing number of sessions coexist in the overlay network, it is much more important to balance the traffic among different links. As we can see in Fig. 4(b) the curves for the B algorithm and the A algorithm grows slowly. It means these two algorithms are good at accepting a lot of sessions at the same time.

5.2 Performance on Traffic Balancing

Another performance criterion for these path selection algorithms is how much they can balance their traffic load among their access links. This is important for keeping a low possibility of traffic congestion. In order to measure it accurately, we invent a parameter named balance degree. It equals to the difference of the residual capacity between the least loaded access link and the most loaded access link of a single MSN. The balance degree for an overlay network is obtained by averaging the value among all MSNs. When this value becomes bigger, the overlay network has a less balanced traffic over all its access links. We measure this value as the session number grows gradually. The simulation result is shown in Fig. 5. According to this figure we can infer that the S algorithm is less efficient in traffic balancing.

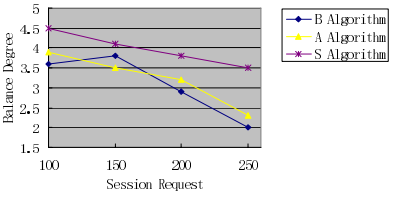


Fig. 5. Balance degree VS session requests

6 Conclusion

In this paper we proposed a multihomed proxy model to analyze the overlay multicast routing problem in a fine granularity so that we can perform the traffic balancing on a link basis rather than on a node basis. We also compared three

path selection algorithms, which do the necessary preparation before the routing process can start. Through simulation results and numeral analysis, we can see that although the alternative path algorithm needs a little more computation than the other two algorithms, it results in a good performance in both tree diameter and traffic balancing.

References

1. S.Y.Shi and J.S. Turner: Routing in overlay multicast networks. *Proc. IEEE INFOCOM*, June 2002.
2. Y.-H. Chu, S. G. Rao, and H. Zhang: A Case for End System Multicast. *Proc. ACM Sigmetrics*, June 2000.
3. S. Shi, J. Turner, and M. Waldvogel: Dimensioning server access bandwidth and multicast routing in overlay networks.*Proceedings of NOSSDAV*, June 2001.
4. Deering, S.: Multicast Routing in Internetworks and Extended LANs. *SIGCOMM Summer 1988 Proceedings*, August 1988.
5. Li , P. Mohapatra: QRON: QoS-aware routing in overlay networks. *IEEE JSAC*, 2003
6. S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller: Construction of an efficient overlay multicast infrastructure for real-time applications. *IEEE INFOCOM*, San Francisco, CA, April 2003

Region-Based Multicast Routing Protocol with Dynamic Address Allocation Scheme in Mobile Ad-Hoc Networks

Backhyun Kim and Iksoo Kim

Department of Information and Telecommunication Engineering, Univ. of Incheon,
177 Towhadong, Namku, Incheon, Korea
{hidesky24, iskim}@incheon.ac.kr

Abstract. Mobile Ad hoc Network (MANET) is a network that mobile nodes can freely and dynamically communicate with each others without the support of any infrastructure. Due to the node's movement, the network topology can be changed randomly and unpredictably. Previously, address as identifier is used for establishing a delivery route. But it is not easy that each mobile node has a unique address due to the lack of centralized address protocol like DHCP. In this paper, we propose region-based dynamic address allocation scheme that can simplify the route establishment and maintenance. Routing is done by a short distance vector algorithm based on the proactive and the flat topology routing protocol. Basic idea of our address allocation scheme is to separate node identity and node address. Node address indicates node's current location and is allotted by its parent node with a cascading address scheme. In evaluation, we examined our proposed scheme in a point of the view of the number of control packets to establish delivery trees and the connecting probability. From simulation results, we confirm that the proposed scheme offers substantially better performance.

Keyword: Ad-hoc, routing, wireless networks, address allocation, multicast.

1 Introduction

A fundamental characteristic of mobile wireless networks is the time variation of the link status between a sender node and a receiver node. Such variation can be due to the node mobility and/or the lack of channel capacity. Thus a scheme to keeps up-to-date routing information is required. In aspect of a route construction mechanism, wireless mobile networks can be categorized as controlled networks and distributed networks. In controlled networks such as wireless LAN and mobile cellular networks, because route establishment sequence is handled by central coordinators, mobile nodes (MNs) can communicate with each other without any route information toward destination node. In distributed networks, since there is no central coordinator, MNs should organize network topology by themselves.

This paper presents a new data delivery technique with a dynamic address allocation scheme that can simplify the route establishment and maintenance by minimizing control packets. In this paper, we use short distance vector algorithm

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 137 – 146, 2006.
© Springer-Verlag Berlin Heidelberg 2006

based on the proactive and the flat topology routing protocol in MANET. Delivery tree construction sequence is started by a MN which wants to receive the data from a source node or another MN. The transmission sphere of MN can be divided into six regions similar to hexagonal plane of mobile cellular network. Basic idea of our address allocation scheme is to separate node identity and node address. Node address indicates node's current location and is allotted by its parent node with a cascading address scheme.

Under the proposed scheme, one of MNs is elected as address allocator (AA). We assume that AA is access point (AP) in IMANET. The coverage area of AA can be divided into six regions. MNs in each region have same address assigned by AA and one of them becomes a root node for each delivery tree. Thus six root nodes can make six different delivery trees. The addresses for MNs 2-hops away from AA are assigned by their root nodes and become their current location identifiers followed by their root nodes' addresses. Therefore, the address for MN n hops away from AA is generated by MN n-1 hops away from AA on delivery route tree. MNs in the same region are classified by node identifications as IP address, MAC address, etc. With proposed address scheme, as the address of destination node indicates the entire multi-hop path from AA to destination, each MN maintains routing information of only neighboring nodes to keep up-to-date route. Neighboring nodes can communicate directly as they are within transmission range of each other.

After finishing tree construction sequence, every MN over its delivery tree becomes either forwarding MN (FwMN) or request MN (ReqMN). FwMN is the node that is not interest in the data being transmitted but should relay the data to the request MN. When receiving data, FwMNs and/or ReqMNs operate in multi-hop fashion with store and forward manner until the data reach the final ReqMN.

The remainder of the paper is organized as follows: The next section we summarize previous work. Proposed multicast scheme is introduced in section 3. In section 4, we present the simulations and analysis of the results. Finally, we give out conclusion in section 5.

2 Previous Work

Multicast allows clients to share data streams being transmitted through one channel, where channel is the unit of network bandwidth needed to transmit one stream [1]-[3]. In multicast, the requests for the same content are delayed for a certain amount of time $\boldsymbol{I_m}$ to serve as many requests as possible with one multicast channel. It is regularly generated at every $\boldsymbol{I_m}$ when there are the requests for the same content within $\boldsymbol{I_m}$. So, some of clients should wait until a new multicast is generated.

The routing protocols for MANET are split into two categories based on the routing information update mechanism. They are proactive and reactive routing protocols. In proactive method [4]-[6] called table-driven routing protocol, every node maintains the network topology in the form of routing tables by periodically exchanging routing information that are flooded in the whole network. Though this method can achieve up-to-date route information, it consumes much more network bandwidth caused by generating unnecessary traffics. Reactive [7],[8] is called on-demand routing because it executes path finding only when MNs want to

communicate with another one. So in reactive, every node does not need to maintain the whole network topology while proactive routing protocols do. This method can reduce traffics not sending route information periodically and thus increase network throughput. But since it should set up the route before sending the data, service will be delayed until the route setup sequence will be finished. Route construction can be done in either a flat topology or in a hierarchical topology. In flat topology, address for MN is globally unique and route information for updating and maintaining network topology is flooded over whole network. Hierarchical mechanisms split an entire flat topology into a logically hierarchical structure called zone limited within a particular geographical region.

Dynamic address allocation can simplify routing procedure. Thoppian et al. [9] propose dynamic address assignment that allots a unique IP address to a new node joining in MANET. In this scheme, each MN has some IP address block. When a new node (requester) join a MANET, one of the existing MANET nodes (allocator) within communication range of the requester allots the second half of the addresses from its free_ip set to the requester. Though this method guarantees unique IP address assignment under a variety of network conditions, it cannot reduce routing overhead as each node has the same length of address, i.e. 4 bytes in IPv4. Eriksson et al. [10] propose a variable length of dynamic addressing scheme based on a hierarchical binary tree structure of proactive distance vector routing. This scheme separates node identity from node address that indicates the node's current location in the network. If there are n nodes, address length is $\log_2 n$ and average routing table sizes are less than $2\log_2 n$. Node lookup information is distributed in the network as node lookup to find the current address of a node is done by hash function. As address length is proportional to the number of nodes in the network, it produces routing overhead in a dense network and data will be taken along a longer path instead of the shortest route. Chen and Nahrstedt [11] propose address compression scheme to reduce routing overhead with overlay multicast in MANET. This method also separates node identifier from node address called index. A node's index is determined by the application server when it joins the multicast group. Node lookup is done by the application server and then this information is recorded in each node's address lookup table. As this scheme assigns unique index to each node, it is not a proper scheme in dense network.

3 Proposed Scheme

3.1 Operation

Mobile network is generally described in hexagonal cellular structure due to the same distance between any adjacent cells. Proposed mobile wireless networks are operated in the same plane structure for mobile networks but not existing the central coordinators at every cell. Because the cells have been configured by MNs, the cell structure is not fixed as mobile cellular networks. For supporting seamless service using multicast, the wireless networks are consisted of a base node (BN) and a number of mobile nodes (MNs). The BN may be a MN or fixed node as AP in IMANET, and the number of BN can be either one per a basic tree or only one in ad-hoc network. MNs on multicast delivery tree have the role of either forwarding node (FwMN) or request node (ReqMN).

Fig. 1 shows the proposed multicast delivery scheme in flat topology. Route construction is done whenever MNs want to join a multicast. But to keep up-to-date routing information, MNs periodically broadcast connection information about only their neighboring nodes that are 1-hop away. Minimum distance algorithm based on hop counts is used for route selection. From Fig. 2, multicast group member is {1, 2, 3, a, b, d, e, u, v, w, x, y, z} in initial stage. MNs 1-hop away from BN are called root node and there are three root nodes {1, 2, 3}. Each root node has child nodes {a, b, d, e}, {w, x, y, z} and {u, v}, respectively.

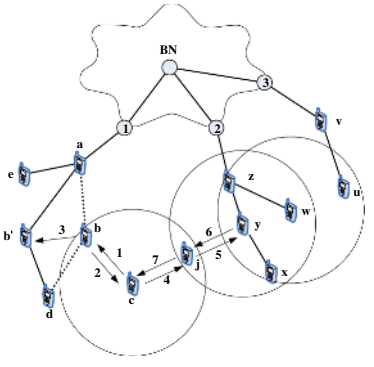


Fig. 1. Example of multicast deliver tree

The multicast delivery tree generated from a specific root node is called *basic tree*. Creating a basic tree is initiated by broadcasting a packet (request packet; REQ) of ReqMN in mobile wireless network. Since some of them receive directly reply packet (REP) from BN, they are the group of the 1- hop MNs, i.e. root nodes. The MNs which receive it directly from 1-hop MNs become 2-hop MNs. Therefore MNs received REP from n-1 hop MNs become n-hop MNs.

In conventional multicast techniques, to join a multicast group, MNs forward REQ until they meet multicast delivery tree. If MN is n-hops away from multicast tree, $2n$ control packets are generated because there are n REQ packets and n REP packets. For example, if MN c wants to join a multicast, it broadcasts REQ to its neighboring nodes (arrow 1 in Fig. 1). When MN b receives REQ, it replies to MN c with REP as it is already a member of multicast (arrow 2). MN is able to know what MNs are in its transmission range. We call them neighboring mobile node (NMN). NMNs of n-hop MN are consisted of the upper MNs (n-1 hop), lower MNs (n+1 hop) and peer MNs (n-hop). NMNs of MN c are {b, j}. Under the proposed scheme, since all MNs broadcast route information periodically, MN c has already known that MN b is a member of multicast. Hence MN c just receives the data generated by MN b, i.e. need not perform step 1 and 2. This can reduce $2n$ control packets compared to conventional schemes mentioned above.

MN y and MN w have NMNs {j, w, x, z} and {u, x, y, z}, respectively. The NMNs of MN y in multicast delivery tree has the same root node MN 2. But MN u of NMNs of MN w is originated from the different root node MN 3. We call this foreign NMN. Let L(x, y) be the communication link between MN x and MN y. When MN b move

to b', the link L(b, c) is broken due to out of transmission range (arrow 3). To reconstruct multicast tree, MN c performs the same sequence mentioned above (arrow 4, 5, 6, 7). In consequence, MN c finds a new route along MN j, y, z, and 2. MN j has joined the multicast as FwMN and two communication links L(j, c) and L(y, j) are established. When MN j receives REQ from MN c, because MN j already knows the existence of the requested multicast, it just receives multicast streams from MN y and forwards them to MN c. Thus under the proposed scheme, tree reconstruction flow is done through step 4 and 7. Because MN c should broadcast REQ to make communication link between MN c and MN y, it may generate numerous control packets. Under the proposed scheme, MN manages connection information about NMNs. Basically MN knows the existence of MNs within its coverage area. This connection information is sent to its NMNs. MN can know the existence of MNs 2-hops away from itself. In above case, MN c sends REQ only to MN j and then MN j forwards the request multicast from MN y to MN c. To establish a link between MN c and MN y, it needs only two control packets that are generated between MN c and MN j. Thus the proposed scheme can reduce the number of control packets in order to establish multicast delivery tree and the traffic can be localized.

3.2 Address Allocation

We proposed an address allocation scheme to identify MNs based on the hexagonal plane structure depicted in Fig. 2. The proposed address allocation scheme is consisted of two part; logical address(LA) and physical address(PA). LA indicates the route from BN to its current locating cell. As the number of MNs in a cell may be more than one, the mechanism to identify among them is needed. For this, we use physical address using link-layer address(MAC address). Fig. 2 shows the location estimation in hexagonal plane where multicast delivery tree is the same Fig. 1. MN has one parent MN and several child MNs. Because one cell is surrounded by 6 cells, 3 bits address is sufficient to identify them. But the 2 adjacent cells next to parent cell become peer cells because they have the same number of hop counts. Thus it is needed only 2 bits to identify cells except root cells in which root cells locate. The MN that becomes the first child MN gets 01 as its address. The second connected MN obtains its address either 01 or 10. If the position of the first connected MN is within the transmission range of the second connected MN, its address is 01. If not, its address is 10.

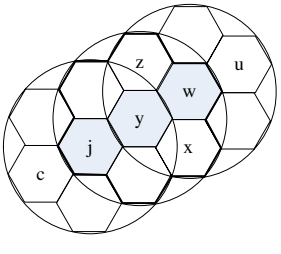


Fig. 2. Hexagonal structure for multicast delivery tree

From Fig. 1 and 2, MN z has two child MNs {y, w}. Initially they have the same address 01. But since MN w know the existence of MN u, the address of MN w is changed to 11 as MN u is a foreign NMN. Thus MN y can know the existence of MNs out of its transmission range. MN y has only one child node MN x. When the link between MN b and MN c is broken due to the movement of MN b, MN j joins the multicast and then becomes child node of MN y. As MN j cannot communicate with MN x, the address of MN j becomes 10. Let the address of root node 2 be 010. The address of MN z becomes 010-01 because root node 2 has only one child node. The addresses of MN w and MN y are 010-01-11 and 010-01-01, respectively. The addresses of MN j and MN x that are 4 hops away from BN are 010-01-01-10 and 010-01-01-01, respectively. The logical address of a MN locating at *i* hop away from BN can be represent a cascade addressing algorithm which makes new address based on received address; the address till *i*-1 hop followed by *i*'th logically connected information.

This paper adopts mapping table to convert logical address into physical IP address. All MNs on the same basic tree have the same root-field to classify basic trees and mapping table. Although logical address needs theoretically only *k*-bit (log_2 *n: n is number of MNs*) in order to assign all MNs, this paper needs more bits for managing NMNs and indicating whether MNs join multicast group or not. The structure of mapping table to manage multicast delivery trees is composed of 4-fields; Multicast-field, Tag bit, logical and physical addresses (see Fig. 3(a)). And logical address indicates the information of basic tree, and it is divided into root and peer-classifying field of MNs. These address allocation is self-configured [6, 11, 12]. The values of root and peer-classifying field for MNs are generated sequentially by BN and the upper MNs, respectively [10].

m bits	1 bit	3bits	2n bit
Multicast (M)	Tag (T)	Root (R)	Peer Classifying Field (PCF)

(a) Address structure

M	T	R	PCF	MAC
10	0	001	MN a : 01	32120abcd
10	1	001	MN b : 01-01	3847ac13
00	1	010	MN j : 00	95031243
10	0	010	MN y : 01-01	12095833

(b) Mapping table of MN c before MN b moves to b'

M	T	R	PCF	MAC
10	1	010	MN j : 01-01-10	95031243
10	0	010	MN y : 01-01	12095833

(c) Mapping table of MN c after MN b moves to b'

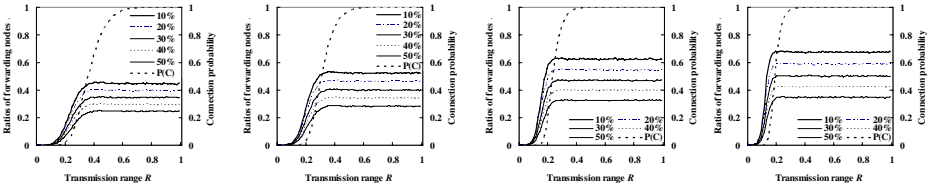
Fig. 3. A mapping table of MN *c* according to a movement of MN *b*

Fig. 3(b) shows mapping table for MN c before MN b moves to b'. Its NMNs are {a, b, j, y}. The m-bits multicast-bit indicates whether the MNs have already joined

the specific multicast or not. In Fig. 3, we assume that the number of m bits is 2. If a NMN has not joined any multicast groups, M bits are represented with the value of 00. Tag indicates whether a NMN or not. Root field represents the address of root node of MNs. If R fields of MNs are same, they are on the same basic tree. Peer classifying field (PCF) represents the connection status and the number of hops counted from base node. From mapping table, PCF of MN 2 is 00. It means that MN 2 is root node addressed 010. In Fig. 3(c), the number of hops for MN j is 4 as PCF of MN c is 6 bits. The address of MNc is changed from 001-01-01-01 to 010-01-01-10-01 due to changing of its root node.

4 Simulation and Analysis

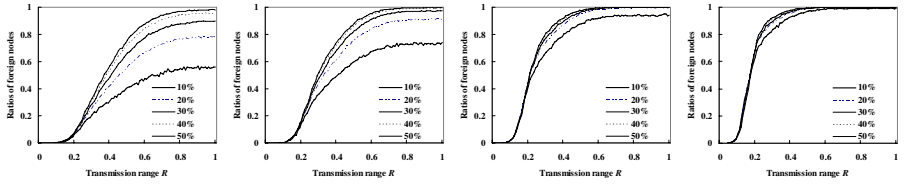
In this section, we show simulation results to demonstrate the benefit of proposed mobile ad hoc network to support robust streaming service, and analyzes on the results of performance using it. We assume that simulation network is created in rectangular planer space with 10 to 50 MNs that are homogeneous and energy-constrained. In the simulation plane based on hexagonal cellular architecture shown in Fig. 2, BN locates at the center of plane. It can stream the contents with directional propagation manner to distinct root nodes. BN can partitions the space into multiple cone areas and selects NMNs in each cone area as root node of each multicast delivery tree. MNs in each cone area forward a copy of the packet to their child MNs. This heuristic ensures that the groups of the subtrees are along approximately the same direction from the BN. The fan-out angles of BN are equal to 60° because the proposed plane is based on hexagonal cellular architecture.



(a) N = 10 (b) N = 15 (c) N = 30 (d) N = 50

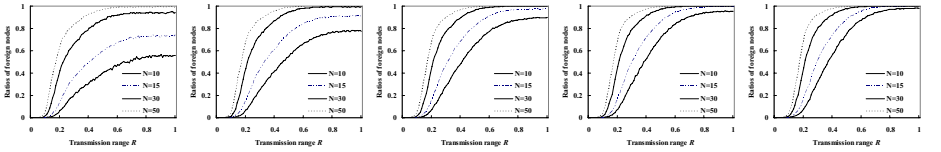
Fig. 4. The ratio of forwarding nodes as the function of request rate from 10% to 50% according to the various numbers of nodes in wireless networks

Fig. 4 shows the ratio of forwarding nodes over multicast delivery trees. The X-axis shows the transmission range *R* while the left Y-axis shows the ratio of forwarding MNs and the right Y-axis shows the connection probability of MTE scheme with fixed transmission range shown in Fig. 2. The simulations are done with transmission range *R* and the request rate in N = 10, 15, 30 and 50, where N is the number of MNs. The request rate is the percentage of the number of ReqMNs over the total number of MNs. The simulation results indicate that the more the request rate increases, the more the number of forwarding MNs is. The convergences of the number of FwMNs occur in shorter transmission range when the number of MNs is larger.



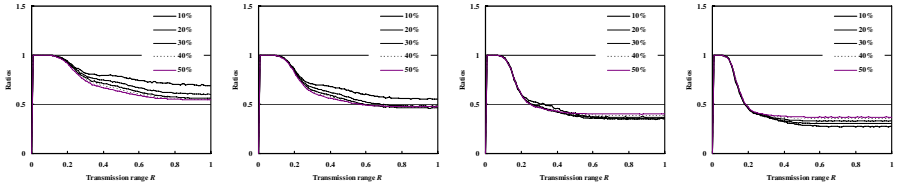
(a) N = 10 (b) N = 15 (c) N = 30 (d) N = 50

Fig. 5. The ratio of foreign nodes as the function of request rate from 10% to 50% according to the various numbers of nodes in wireless networks



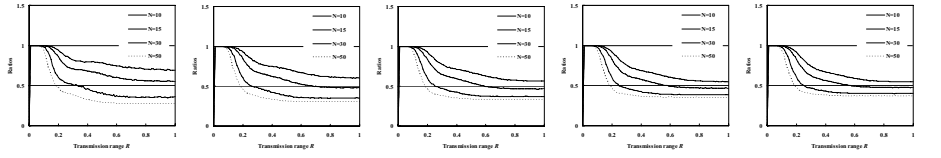
a) Request = 10% (b) Request = 20% (c) Request = 30% (d) Request = 40% (e) Request = 50%

Fig. 6. The ratio of foreign nodes as the function of the number of nodes 10, 15, 30 and 50, according to the various numbers of request rates from 10% to 50%



(a) N = 10 (b) N = 15 (c) N = 30 (d) N = 50

Fig. 7. The packet ratio compared with MTE as the function of the number of nodes 10, 15, 30 and 50, according to the various numbers of request rates from 10% to 50%



(a) Request = 10% (b) Request = 20% (c) Request = 30% (d) Request = 40% (e) Request = 50%

Fig. 8. The packet ratio compared with MTE as the function of the request rates from 10% to 50%, according to the various numbers of nodes in wireless networks

If multicast delivery tree has been broken, MNs on trees should find another one to achieve seamless multimedia service. Paths broken occur in between root node and its child nodes. Thus there is need to find a path generated by another root node, i.e. find foreign MNs. Fig. 5 shows the ratio of foreign MN as the function of the requests

rates where the number of MNs is 10, 15, 30 and 50. The X-axis shows the transmission range *R* while the left Y-axis shows the ratio of foreign MNs. Simulation results represent that there are less difference of ratio if the number of MNs is much larger. Because the high request rate needs much more forwarding MNs, the ratio of foreign nodes is increased if the number of forwarding nodes is increased. Fig. 6 shows the ratio of foreign MNs over the total number of multicast member MNs where the range of the request rate is 10% to 50%. The X-axis shows the transmission range *R* while the left Y-axis shows the ratio of foreign MNs.

Fig. 7 shows the comparison between conventional scheme and proposed scheme. The X-axis shows the transmission range *R* while the left Y-axis shows the ratio of the number of control packets for proposed scheme over the number of control packets for conventional multicast scheme with MTE. From the result, proposed scheme saves the total network bandwidth consumption compared to the conventional scheme all the time. Fig. 8 shows the ratio in a point of view of the total number of MNs in wireless networks based on the various request rate 10% to 50%. From the simulation results, the proposed scheme can reduce the required control packet to establish multicast delivery tree.

5 Conclusion

In MANET, the mobility of nodes results in frequent path breaks, packet collisions, transient loops, stale routing information, and difficulty in resource reservation. Thus streaming multimedia is not easy to implement due to the frequent path broken. In this paper, we proposed a content delivery scheme to achieve the robust streaming of multimedia contents. Under the proposed scheme, MN broadcasts information about its neighboring MNs periodically. MN makes a routing table with received information. MN can know the existence of MNs 2-hops away from itself because the broadcasting packet includes information only about MNs 1-hop away. Thus it can reduce the number of control packets to establish multicast delivery tree. To maintain multicast route, we proposed new address allocation scheme with cascading method based on hexagonal plane. The address of MN is based on the relation between itself and other MNs. MN can split their outer area into multiple cells and establish alternative route by selecting another MN having the different root node when the route is broken. Cascading address make MNs calculate the distance from BN and estimate the existence possibility of the alternative route toward BN. In evaluation, we examined our proposed scheme in a point of the view of the total number of control packets and the connecting probability as well as the impact of the number of MNs in mobile wireless networks. The simulation results indicate that the number of MN and the request rate are the critical performance factor in order to minimize the network bandwidth consumption.

Acknowledgement

This work was supported by the University of Incheon Research Grant in 2006

References

[1] T. Ozaki, J. B. Kim and T. Suda, "Bandwidth efficient multicast routing protocol for ad hoc networks," Proc. of IEEE ICCCN 1999, pp.10-17, Oct, 1999
[2] S. H. Bae, S. J. Lee, W. Su, and M. Gerla, " The design, implementation and performance evaluation of the on-demand multicast routing protocol in multihop wireless networks," IEEE Network, pp.70-77, Feb, 2000
[3] E. M. Royer and C. E. Perkins, "Multicast operation of the ad hoc on-demand distance vector routing protocol," Proc, of MobiCom, pp.207-218, 1999
[4] C.E. Perkins and P.Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, "Computer Commun. Review, Vol.24, no.4, pp.234-244, Oct.1994
[5] T. Clausen et al., "Optimized link state routing protocol," IETF RFC 3626, Oct. 2003
[6] T.W. Chen and M. Gerla, "Global state routing: a new routing scheme for ad-hoc wireless networks," Proc. of IEEE ICC, pp.171-175, June 1998
[7] D.B. Johnson and D.A. Maltz, "Dynamic source routing in ad hoc wireless networks in mobile computing," T. Imielinkski and H. Korth, Eds., Kluwer Academic Publishers, Dordrecht, pp.153-181, 1996
[8] C.E. Perkins, E.M Belding-Royer, and S.R. Das, "Ad hoc on-demand distance vector (AODV) routing," IETF RFC 3561, July 2003\
[9] M.R. Thoppian and R. Prakash, " A distributed protocol for dynamic address assignment in mobile ad hoc networks," IEEE Trans. on mobile computing, Vol.5, No.1, pp.4-19, Jan. 2006
[10] J. Eriksson, M. Faloutsos, and S. Krishnamurthy, "Scalable ad hoc routing: the case for dynamic addressing," In Proc. Of IEEE Infocom 2004, 2004
[11] K. Chen and K. Nahrstedt, "Effective location-guided overlay multicast in mobile ad hoc networks," International Journal of Wireless and Mobile Computing (IJWMC), Special Issue on Group Communications in AD Hoc Networks, Vol. 3, 2005
[12] W. Heinzelman, J. Kulik, and H. Balakrishnan, "Energy-efficient communication protocol for wireless microsensor networks", In Proc. of the Hawaii Int. Conf. on System Sciences, vol.8, pp.3005-3014, Jan. 2000
[13] D.B. Johnson and D.A. Maltz, "Protocol for adaptive wireless and mobile networking," IEEE Personal Commun., pp.34-42, Feb. 1996
[14] M.S. Corson, J.P. Maker, and J.H. Cernicione, "Internet-based mobile ad hoc networking," IEEE Internet Computing, Vol.3, no.4, pp. 63-70, 1999
[15] M. Conti and S. Giordano, "Mobile ad hoc networking," Cluster Computing Journal, Vol.5, No.2, April 2002
[16] P. Gupta and P.R. Kumar, "Critical power for asymptotic connectivity in wireless networks," A Volume in Honour of W.H. Fleming in Stochastic Analysis, Control, Optimization and Applications, 1998
[17] P. Panchapakesan and D. Manjunath, "On the transmission range in dense ad hoc radio networks," Int. Conf. on Signal Processing and Commun. SPCOM, IISc, Bangalore, July 2001

A Novel Approach for Topology-Aware Overlay Multicasting

Xiao Chen, Huagang Shao, and Weinong Wang

Regional Network Center of East China,
Department of Computer Science and Technology,
Shanghai Jiao Tong University, China
{shawn, hgshao, wnwang}@sjtu.edu.cn

Abstract. Most existing overlay multicast approaches avoid to consider any network layer support no matter whether it is available or not. This design principle greatly increases the complexity of the routing algorithms and makes the overlay topologies incompatible with the underlying network. To address these issues, we propose TOMIMN as a novel overlay multicast approach, which exploits the cooperation between end-hosts and IP multicast routers to construct a topology-aware overlay tree. Through a little modification to PIM-SM, a multicast router is able to receive registration from nearby group members and redirect passing-by join requests to them. Due to the multicast router's support, TOMIMN organizes its group members into an overlay multicast tree efficiently, which matches the physical network topology well.

1 Introduction

Most of the current overlay multicast solutions (e.g. Narada[3], Nice[2]) share some common drawbacks. First, they can't adapt the overlay to the physical network very well. Thus it is hard to guarantee the cost-efficiency of data transmission in all cases. Second, they force end-hosts to assume too much responsibility in overlay construction, which makes the protocol too complicated to serve as a general platform for other applications. Last, since the recovery from overlay partition can be very slow, it is hard to maintain a stable performance in case of node failures. All these issues result from a common design principle. That is to remain the network layer as simple as possible and refuse to consider any support from the network no matter whether it is available or not.

While advocates of overlay multicast try to assume a simplest network layer, IP multicast has won a position among most enterprise-level networks due to its efficiency and effectiveness in enabling small-area group communications. Through almost 20 years' development, most of the IP multicast techniques have been standardized as protocols (e.g., PIM-SM[8]) and been supported as mandatory functions by most network equipment vendors. Corporations are willing to deploy multicast-capable routers in their internal networks since IP multicast provides a better solution to a lot of valuable applications such as network conference and content distribution. As an increasing number of multicast-enabled networks are

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 147–158, 2006.
© Springer-Verlag Berlin Heidelberg 2006

emerging on the Internet, the lack of hardware support from network layer is no longer a persuasive argument against the IP multicast mechanism.

The true weakness of IP multicast is its inadequacy in dealing with large-scale group communications in the Internet environment. This is due to the following design flaws. First, since class D addresses can't be aggregated by their prefixes, a multicast router in a network core can be overwhelmed when dealing with thousands of groups simultaneously. Second, address collision is hard to avoid when a large number of groups coexist in the Internet. Last, there is no viable commercial model for IP multicast. This fatal factor makes the Internet service providers (ISPs) reluctant to open IP multicast as a public service.

Our contribution in this paper is to address all above issues of different multicast techniques by proposing a topology-aware overlay multicast approach, which is supported by IP multicast networks. The key point of our approach is to adopt the overlay network for actual data transmission while exploiting IP multicast routers to optimize the overlay construction. This method has several advantages over traditional overlay multicast schemes. With support of the multicast-enabled network layer, the task of overlay topology optimization can be drastically simplified. The complexity of the dynamic membership management is also reduced to a large extent. Meanwhile, end-host based multicast eliminates most of the problems related to IP multicast. Since the multicast routers may be deployed incrementally in our approach, the overlay can easily scale up to support large-sized group communications among heterogenous networks, where not all routers are multicast-capable. Besides the scalability issue and the address collision, overlay multicast also provides a viable solution to the commercial issue, which is difficult in IP multicast. Therefore the two multicast techniques, which seem to contradict at a first glance, are merged to produce an efficient multicast framework in this paper. Hereinafter, we name it by TOMIMN, which stands for *Topology-aware Overlay Multicast over IP Multicast Network*.

The rest of the paper is organized as follows: Section 2 reviews some basic ideas that previous works adopt in dealing with the overlay topology issue. In Sect.3, we define the network model for TOMIMN and clarify the objectives we want to achieve. We also analyze the mechanism of IP multicast in this section. The detailed protocol is explained in Sect.4, which describes our modification to the PIM-SM protocol and the tree construction algorithm. The performance of TOMIMN is evaluated through simulations in Sect.5. Finally we draw conclusions in Sect.6.

2 Related Works

While a variety of overlay multicast approaches have emerged in the past few years, most of them adopt a similar mechanism to adapt their overlay topology to the underlying network. As a typical instance, Narada[3] estimates the latency and availabe bandwidth of each overlay link by means of *ping* and passive monitoring respectively. Then it runs a distance vector protocol on each node to construct an overlay network according to its end-to-end measurement.

While other overlay multicast protocols may differ in their specific application-layer routing algorithms or measurement approaches, they share a common point with Narada. That is to regard the physical network as a black box and base their overlay construction solely on the end-to-end network measurement. As it is hard to derive the underlying network topology from the measured data, the result overlay topology can be very inefficient.

TAG[7] is a famous topology-aware overlay multicast approach, which has a similar objective as TOMIMN has. It is different from traditional overlay schemes in that it exploits underlying network topology data to construct its distribution tree. More specifically, it uses *traceroute* to discover the route from the source to each member and then depends on these information to select parents for new joiners. Since *traceroute* reads the route information through ICMP echoes, TAG is effectively supported by the network layer in its tree construction, which is similar to our approach. However, TAG doesn't perform as well as TOMIMN in the tree optimization as the simulation results indicate. In addition, the low efficiency of *traceroute* in obtaining the route information imposes a big control overhead on TAG's execution.

A more imaginative idea is to assume special primitives on routers to make the overlay more adaptive to the physical network. The network layer support proposed in [6] just belongs to this kind. The author devises two novel primitives to construct the overlay multicast tree efficiently. *Packet reflection* allows end-hosts to benefit from advantageous position of routers for moving and duplicating packets. *Path painting* allows end-hosts to use routers to learn enough about the network to build efficient overlay topologies. While such primitives seem reasonable at a first glance, they make the network layer even more complex than before, which contradicts the original intention of overlay multicast. In contrast, TOMIMN only exploits available network support from IP multicast routers so that it is more practical in the current Internet context.

3 Network Models

3.1 Physical Network Model

Without loss of generality, we model the underlying network in question by a directed graph G(V, E) with the vertices representing routers or hosts and the edges standing for links. A host is attached to only one router while a router's node degree is more than one. Each edge is assigned a weight to indicate the link's cost or latency. We assume that each node here belongs to an autonomous system (AS) that is either a multicast domain or a unicast domain. All routers of a multicast domain support intra-domain IP multicast. To make the following description easier, we now define some concepts, which will be used throughout the paper.

Definition 1. *A path from node A to node B, denoted by* $P(A, B)$*, is a sequence of routers comprising a shortest path from A to B according to the underlying unicast routing protocol.* $P(A, S)$ *is also called a request path of node A if node*

S is a data source. Since a unique data source is usually presumed, the request path of A is also denoted by RPath(A).

Definition 2. *Node B is close to the request path of node A if at least one node of RPath(A) is within a certain distance from node B and they are located in the same multicast domain. This distance is also called a capture range, which is determined by the physical network settings around node B.*

An outline of the physical network model is shown in Fig. 1. In this figure, all router nodes are marked with capital letters while the gray nodes represent end hosts. Suppose that node s is the unique source and all other hosts are receivers of the same group. We use the dotted lines to indicate the request paths of $RPath(a)$ and $RPath(b)$, which originate from a and b respectively and point to node s. If the capture range is just one hop, only c and g are close to $RPath(b)$. Since $e1$ and $e2$ are located in a unicast domain, they are not close to $RPath(b)$ even if they are just one hop away from the router E.

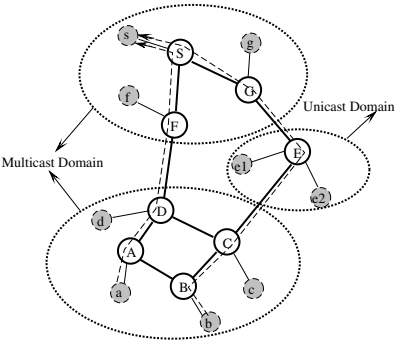


Fig. 1. Physical network model

3.2 Overlay Network Model

Overlay multicasting is to send one copy of the data from a source node to several receivers and then retransmit it from those nodes to other receivers. The process goes on and on until all receiver nodes get the multicast data. In tree-based overlay multicast, the protocol will construct a tree, which is rooted at the source and covers all receivers. Note that this tree only consists of end-hosts, which are members of the multicast session. So we call such a tree an overlay tree, in contrast with any other multicast tree, which may contain router nodes. An edge of the overlay tree is called an overlay link, which amounts to the shortest unicast path between two overlay nodes. Our task is to choose the right set of overlay links to connect the members into a tree.

From the perspective of a protocol designer, we have to take the following objectives into account.

1. TOMIMN should be capable to deal with dynamic membership. This is a big obstacle for most overlay routing protocols because without multicast-enabled routers, the arrival or departure of a single member may make the topology of the whole overlay tree change drastically.
2. TOMIMN must be efficient in the construction of the overlay tree. Since the algorithm builds the tree incrementally, it has to be executed in a short time. Otherwise, the continuous join or leave requests will bring about a big burden of computation.
3. The overlay tree should be topology-aware. To avoid unnecessary detour paths among the overlay nodes, the protocol needs to make the overlay tree mimic the IP multicast tree as much as possible so that the data transmission can be done in a cost-efficient way.

3.3 Introduction to PIM-SM

We choose the famous PIM-SM as our prototype of the multicast protocol. For a better understanding of the role a multicast router plays in the TOMIMN, we shall first analyze the key elements that make a PIM-SM router different from a generic unicast router.

Multicast Forwarding Entry. From a multicast router's perspective, the task of any routing protocol is to build a routing table with forwarding entries for all groups. A forwarding entry for the PIM-SM multicasting is denoted by a quaternary tuple of (S,G,*iif*,*oif*), which applies to all packets originated from source S and destined for group G. The other two fields, namely *iif* and *oif*, refer to input interface list and output interface list respectively. The construction of such a forwarding entry and its usage is explained as follows.

Control Path. For simplicity, we only consider the source specific multicast in PIM-SM. Here PIM-SM routers have to deal with two protocols in order to create the forwarding entries for a (S,G) pair. First, the designated router of the receiver will receive an IGMPv3[1] source specific membership report from the host, which carries the (S,G) information. Then it turns this membership report into a PIM-SM join request and unicasts it towards the source S. All routers receiving this request should build an entry like (S,G,*iif*,*oif*). The input interface for receiving the IGMP membership report or the PIM-SM join request is added to *oif* while the output interface for sending the join request is added to *iif*. By this way, the multicast forwarding entry is installed on all PIM-SM routers involved in the group session, which form a shortest path tree from the sender to all receivers.

Data Path. Once the shortest path tree is established, the forwarding entry comes into play for packet multicasting. Whenever a router on the distribution tree receives a multicast packet sent from source S to group G, it will first check whether the input interface of this packet is exactly the one in *iif*. If the packet does come in through the correct interface, a copy of it is sent out through each interface listed in *oif*. The forwarding process is done in a top-down manner from the source's designated router to all leaf routers on the tree.

In summary, the speciality of a PIM-SM router is twofold. From the control path's perspective, it is responsible to accept membership registration originated from the receivers. On the other hand, the router has to multicast packets as its key function on the data path. These two features lend them perfectly to the construction of our topology-aware overlay distribution tree. In the following section, we will describe how these functions can be exploited by TOMIMN after a slight modification.

4 Details of TOMIMN

4.1 Multicasting Join Requests

Multicast-capable routers play an important role in our overlay tree construction. Briefly speaking, TOMIMN depends on those routers on the new joiner's request path to multicast the join request towards their respective neighbor members so that the new member can find a suitable parent from the tree quickly. While a PIM-SM router is totally qualified to assume such a task from a functional perspective, no existing protocol is designed for this purpose. Therefore we need to introduce some modifications to PIM-SM so that the routers will behave as we expect them to.

Multicast Forwarding Entry. One basic function of the multicast forwarding entry is to determine whether a packet belongs to a specific group so that it can be multicasted accordingly. Unlike the situation in source specific IP multicast, where all data packets of the same group have a common source-destination pair like (S,G), the join requests for the same group session originate from different joiners and thus have different source addresses. Since TOMIMN is designed for single source multicast, we will aggregate the join requests of the same session by their destination address/port pair like (S,P). S is in fact the address of the overlay tree's root. The inclusion of the port number P allows for multiple overlay trees rooted at the same node. Therefore the multicast forwarding entry for multicasting the join requests is a ternary tuple like (S, P, *oif*). The field *oif* has a similar meaning as we described previously. Now we are going to explain the installation and application of these entries on routers.

Membership Registration. In order to capture the wanted messages on nearby routers, an in-tree member M sends a membership report toward the tree root with a limited value of TTL. Due to the small TTL, only those routers close to M will receive it. Once a router R has received such a report, which indicates a (S, P) pair, it will initiate a forwarding entry building process, which is described in detail by the pseudo-code routine in Fig. 2.

Forwarding Join Request. The process of multicasting join requests is straightforward once the forwarding entry is calculated as mentioned above. When a join request destined for the session (S, P) arrives at a router where a corresponding forwarding entry is installed, a copy of it is sent through each of the interfaces

```
Procedure ReceiveReportFor (Address M, Address S, Port P)
   //M-Address of the sender of the report
   //S-Address of the tree root
   //P-Port number identifying the group session
   Var
     Entry e;
     //e-Multicast forwarding entry;
   Begin
     If RoutingTable.includeEntryOf(S, P) Then
        e = RoutingTable.fetchEntry(S, P);
        e.addToOif(InterfaceTo(M));
     ElseIf InterfaceTo(M)!=InterfaceTo(S) Then
              e = new Entry (S, P);
              e.addToOif(InterfaceTo(M));
              e.addToOif(InterfaceTo(S));
              RoutingTable.addEntry(e);
     EndIf
   End.
```

Fig. 2. Algorithm for building forwarding entries

in *oif* except the input interface of the request. If the *oif* field only has two interfaces and the request is received from one of them, then the transmission will reduce to a unicast forwarding. This stipulation ensures that all members registered at the routers will receive exactly one copy of the request during the multicasting. A whole process from the membership registration to the join request multicasting is illustrated in Fig. 3.

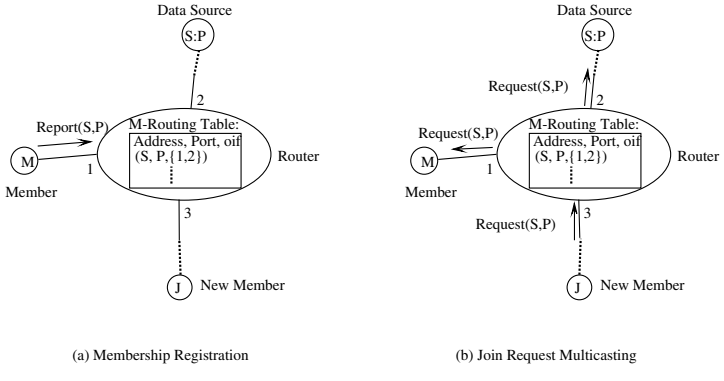


Fig. 3. How to multicast join requests

4.2 Tree Construction and Maintaining

While the modified multicast protocol only focuses on redirecting the join requests of new group members, it has a significant influence on the whole process

of tree construction and maintaining. The support from multicast routers not only simplifies the member join operation but also facilitates the optimization of the tree topology under a dynamic membership. Furthermore, our approach helps to smooth the fluctuation caused by sudden departures of intermediate overlay nodes, which is a tough job for end-hosts alone. Among all these tasks, we should first introduce how these multicast routers help a new member to find a suitable parent in the overlay tree.

Member Joins. Suppose that each new member J knows the address/port pair (S,P) of the data source before its participation. Then a request packet is generated with (S, P) as its destination address/port pair. As we mentioned above, if the join request traverses the capture ranges of other in-tree members, the data source S along with those members close to the request path will be informed of this new joiner. Each of them will respond to J as long as they have sufficient residual fan-out degrees to accept a new child in the overlay tree. Each response indicates a candidate parent. The next step is to choose a best one from them as J's formal parent in the tree.

The specific criteria for choosing a best parent node could vary from one application to another. For simplicity, we order that the candidate closest to the joiner become its parent. If multiple candidates are qualified, the one closest to the data source is selected. This can be done by using tools like *ping* or *traceroute*. Meanwhile, a list of the candidate parents is kept for later use.

Topology Adjustment. As far as tree-based multicasting is concerned, a key reason for topology adjustment is that some in-tree members would like to be connected to newcomers rather than their current parents. This idea is explained in Fig. 4. In this figure, node d is close to the request path of node a. However, node a joins the tree before node d, which leads to a tree structure like Fig. 4(b). While the tree in Fig. 4(d) is more cost-efficient in terms of bandwidth consumption, the join process of d doesn't provides a mechanism to inform the downstream members of the upstream newcomers.

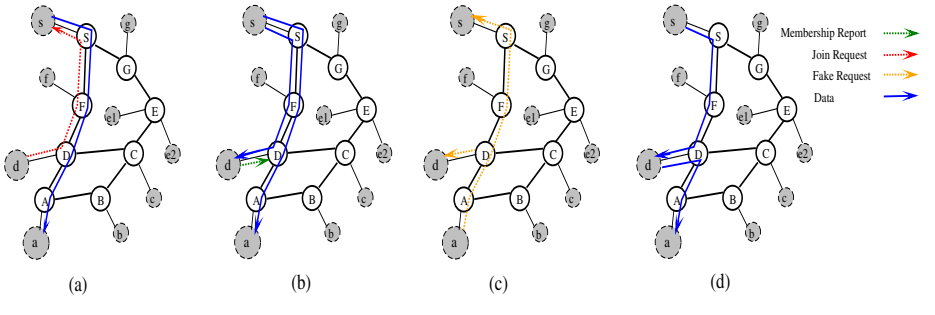


Fig. 4. Topology adjustment

Our solution is to have downstream in-tree members send a fake join request towards the data source periodically. The fake request has the same destination address/port pair as a true one has. Since the newcomer d is close to the request path of node a, it can capture this fake request as shown in Fig. 4(c). Except the newcomer, other members will not respond to a fake request of a familiar sender. Then the tree could be transformed like Fig. 4(d) for a better shape.

Graceful Departure. If a parent node notifies its direct children before it quits the group, each child will follow a query-and-rejoin policy to find a new parent. First, it sends a fake join request as we described previously to discover any new emerging upstream members. If there are replies, the child updates its candidate parent list and pick up a best one from it to contact. The leaving parent doesn't break its transmission until all children have switched to their new parents successfully. When no child listens to it any more, the leaving parent quits the group gracefully.

Partition Recovery. If a failure has occurred, the abandoned children follow a join-and-adjust strategy. In other words, the child first tries to make a connection with the best candidate parent in its current list to join the group. Once the overlay link is established, an adjustment process goes on to optimize the overlay topology after the node failure. Due to the candidate parent list, an abandoned child node can find the new parent immediately while still maintaining a good performance of the overlay transmission. Therefore the impact of node failures is reduced to the minimum.

4.3 Commercial Model

In addition to the efficiency in data transmission, TOMIMN also provides a viable commercial model that facilitates its development in today's Internet. Suppose that ESPN is going to broadcast an NBA game live to the whole world on the Internet using TOMIMN. To provide a satisfactory QoS level to the world-wide audience, ESPN can just rent local hosts as its relay stations to connect with its local subscribers. Since the image signal is transmitted in the name of ESPN, it can provide a worldwide service at a very low cost. On the other hand, the ISPs are encouraged to deploy more multicast routers in their ASes so that they can attract more subscribers for their Internet content providers(ICPs) like ESPN. Due to the potential commercial benefit, both the ICPs and the ISPs are willing to develop TOMIMN as a platform to capture new users for their value-added services.

5 Evaluation

In this section, we evaluate the TOMIMN protocol using simulations. As the multicast routers help us to achieve the first two objectives mentioned in Sect.3.2, the main purpose of the simulations here is to investigate how much a TOMIMN

tree matches the topology of the underlying network. For this purpose, we measure the tree cost in our simulations for a quantified result.

Two benchmark protocols we adopt for comparison with TOMIMN are PIM-SM and TAG. As we introduced in Sect.3.3, we consider the former as a lower-bound benchmark. We also choose TAG for comparison because it is similar to TOMIMN in its design intention.

5.1 Simulation Setup

In order to generate a topology to mimic the Internet for our experiments, we adopt a modified version of the Waxman approach[9]: First, we place a certain amount of nodes randomly on a rectangular area and divide it into several equal-sized regions. All nodes falling into the same region stand for routers belonging to the same domain. Then we build intra-domain links with a probability that is given by the function $P(u, v) = \beta\ exp\ (-d(u, v)/\alpha L)$, where $d(u, v)$ is the distance from u to v, L is the maximum distance between any two nodes of the same domain, and $0 < \beta \leq 1$, $0 < \alpha \leq 1$. Larger values of β result in graphs with higher link densities, while small values of α increase the density of short links relative to longer ones. The first link building process continues until all nodes within the same domain are included into a single connected graph. Next we pick out some gateway nodes from each domain and connect them with inter-domain links in a similar way. The second link building process is finished when no domain is isolated from others. The fan-out degree of each node in the final graph has an upper bound of 6. The cost and delay of each link is set to the same value, which varies from 1 to 5 for intra-domain links and from 6 to 10 for inter-domain links.

To evaluate the performance of our protocol in different network environments, we repeat our simulations in two different scenarios:

1. Physical links have symmetric cost.
2. Physical links have asymmetric cost.

5.2 Simulation Results and Discussions

In the first experiment, we assume that all domains support IP multicast. Whenever a node joins the group in TOMIMN, it will broadcast its membership report throughout its domain to capture other join requests passing by. We change the group size from 20 to 100. Fig. 5 shows the tree cost for two scenarios.

From this figure, we can infer that the performance of TOMIMN is better than TAG and approaches that of PIM as the membership increases. We give the reasons for this result from two aspects. First, as an increasing number of nodes join the group in TOMIMN, it is much easier for a new member to find a parent in its close neighborhood. In other words, the extra cost for accepting a new member into the group decreases when the membership increases. Second, the overlay of TAG is not adapted to the underlying network topology in all situations. Although the parent and its new child share a longest segment on the

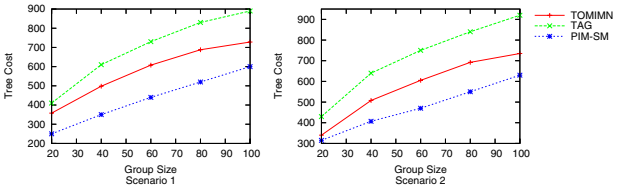


Fig. 5. Group size VS tree cost

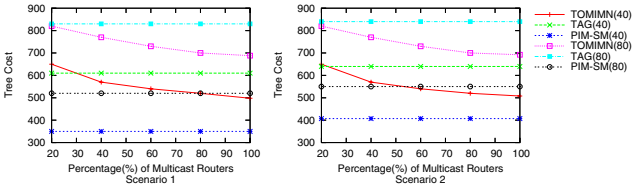


Fig. 6. Percentage of multicast domains VS tree cost

routes from the source to each of them in TAG, it is still possible that the overlay link between these two nodes is longer than other alternatives. Thus TAG is less aggressive than TOMIMN in saving the tree cost. By comparing the output of two scenarios, we find out that TOMIMN is less sensitive to asymmetric links than the other two protocols. Since the cost measurement is always done from upstream nodes to downstream nodes, the asymmetric links don't affect our protocol greatly.

In the second experiment, we assume that only a part of domains support IP multicast. In other words, a member node can't broadcast its membership report around to capture join requests unless it is located in a multicast domain. It is much closer to the situation of the real Internet. We select two fixed group sizes and adjust the percentage of multicast domains from 20% to 100% to study TOMIMN's performance in different environments. The growing percentage of multicast domains is transparent to other two protocols in their simulations. From Fig. 6, we find that the output of TOMIMN is still acceptable even when the multicast domains account for a small percentage in the underlying network.

6 Conclusion

In this paper, we propose TOMIMN as a novel overlay multicast approach, which exploits the support from IP multicast routers to construct a topology-aware overlay tree. Compared with IP multicast and other overlay multicast schemes, our approach has the following notable advantages :

1. A TOMIMN tree is compatible with the physical network. With the help of multicast routers, inefficient branches in a multicast tree are eliminated

2. TOMIMN makes the membership management much easier in overlay multicast. In other words, TOMIMN can deal with member joins and leaves efficiently.
3. TOMIMN encourages ISPs to provide better network layer support. Since the multicast routers can capture consumers' requests of various network services, ISPs are willing to build multicast domains for expectable commercial benefits.

References

1. Cain, et. al.: Internet Group Management Protocol, Version 3, RFC 3376, October 2002
2. S. Banerjee, B. Bhattacharjee, and C. Kommareddy: Scalable application layer multicast. *Proc. ACM Sigcomm*, Aug. 2002
3. Y.-H. Chu, S. G. Rao, and H. Zhang: A Case for End System Multicast. *IEEE J. Select. Areas Commun*, VOL. 20, NO. 8, OCTOBER 2002.
4. Deering, S.: Multicast Routing in Internetworks and Extended LANs. *SIGCOMM Summer 1988 Proceedings*, August 1988.
5. D. Waitzman , C. Partridge , S. Deering: Distance Vector Multicast Routing Protocol , RFC 1075, November 1988
6. John Jannotti: Network Layer Support for Overlay Networks *IEEE OPENARCH*, 2005
7. Kwon, Minseok, and Sonia Fahmy: Path-aware overlay multicast *Computer Networks 2005(47)*, pp. 23–45.
8. Estrin, et. al.: Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification", RFC 2362, June 1998
9. B.M. Waxman: Routing of multipoint connections.*IEEE J. Select. Areas Commun.*, vol.6, no. 9, Dec. 1988.

Limitations of Fixed Timers for Wireless Links

George Xylomenos

Mobile Multimedia Laboratory
Department of Informatics
Athens University of Economics and Business
Patision 76, Athens 104 34, Greece
xgeorge@aueb.gr

Abstract. Traditional reliable link layer protocols set their fixed retransmission timers under the assumption that they operate in isolation over the link. Emerging wireless networks however allow multiple link layer sessions to dynamically share the link. To assess the impact of this development, we examine the performance of Web Browsing over a Selective Repeat protocol with fixed retransmission timers, showing that the optimal retransmission timer values depend on the level of contention. We therefore propose an adaptive Selective Repeat protocol that modifies its retransmission timers based on prevailing conditions. Our measurements show that this adaptive scheme provides excellent Web Browsing performance regardless of the level of contention, under two very different wireless error models.

1 Introduction

Wireless networks are increasingly becoming an integral part of the Internet, especially in the role of access networks providing untethered connectivity to users. It is well known however that the error prone nature of wireless links degrades the performance of applications such as Web Browsing [1]. Reliable link layer protocols have been proposed as a way to hide the deficiencies of wireless links, thus improving the performance of the higher layer protocols and applications used on the Internet; the results reported in the literature show that reliable link layers do indeed offer dramatic performance improvements [2].

While traditional link layer protocols assume that they operate in isolation over the underlying link when setting their operating parameters, such as retransmission timers, emerging wireless networks allow multiple users and/or applications to dynamically share the link. This is most evident in the *Universal Mobile Telecommunications System* (UMTS) where a single physical channel is shared among independent link layer sessions employed by different users and/or applications. Since different applications have different requirements in terms of reliability and delay, when many applications share the same wireless link, different link layer protocols should expect to co-exist over it. While the sharing of a wireless link by independent link layer sessions also introduces fairness issues [2], in this paper we are only concerned with the interplay between link sharing and retransmission timer values and its effect on application performance.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 159–170, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The outline of this paper is as follows. In Sect. 2 we provide background on Internet protocol and application performance over wireless links and discuss related work. Section 3 describes our simulation setup for the performance evaluation that follows. In Sect. 4 we describe the fixed Selective Repeat protocol used in this paper and discuss its performance with Web Browsing. Motivated by these results, in Sect. 5 we present an adaptive Selective Repeat protocol and evaluate its performance with Web Browsing against its fixed counterpart.

2 Background and Related Work

The heart of the Internet, the *Internet Protocol* (IP), offers an unreliable packet delivery service: packets may be lost, reordered or duplicated. Many real-time applications use the *User Datagram Protocol* (UDP) for direct access to this service, handling error, flow and congestion control themselves. Most other applications prefer delegating these tasks to the *Transport Control Protocol* (TCP) which offers a reliable byte stream service. TCP segments the application data stream into IP packets at the sender and reassembles it at the receiver. The receiver generates *acknowledgments* (ACKs) for segments received in sequence, returning duplicate ACKs for out of sequence segments. The sender retransmits the next unacknowledged segment either on receiving 3 duplicate ACKs or when a retransmission timer expires before an ACK is received.

Due to the high reliability of wired links, TCP assumes that all losses are due to congestion, thus after a loss it abruptly reduces its transmission rate to relieve congestion and then gradually increases it to probe the network. Unfortunately, losses due to wireless errors are also interpreted as congestion, causing TCP to dramatically reduce its transmission rate [1]. Many researchers have proposed TCP modifications to improve its performance over wireless links, but they all have two drawbacks: they require modifications to end hosts throughout the Internet and they can only retransmit lost data on an end-to-end basis.

Another approach is to employ a reliable link layer protocol over the wireless link so as to hide wireless errors from TCP. An early proposal customized to TCP *snoops* inside the packets of each TCP stream at the access point bridging the wired and wireless parts of the path and retransmits lost segments when duplicate ACKs arrive, hiding them from the sender to avoid end-to-end recovery [3]. Later work shows that the performance of TCP applications can be enhanced with standard reliable link layer protocols, without making the link layer TCP aware [2]. Avoiding TCP awareness has many advantages, such as compatibility with encrypted IP payloads which hide TCP headers from the link layer [4].

An important issue with reliable link layer protocols is that not *all* Internet applications require their services. While TCP applications are well suited to them, delay sensitive UDP applications often prefer faster, albeit limited, error recovery. Reliable link layer sessions should therefore expect to co-exist with other link layers over the same wireless link. This causes the bandwidth available to the reliable link layer protocol, and therefore its effective *Round Trip Time* (RTT), to vary, leading to a problem when setting retransmission timers:

they should be higher than the RTT, to prevent premature retransmissions, but not too high, to prevent the protocol from stalling until a timeout occurs. A TCP aware link layer sets its retransmission timers dynamically by mimicking TCP retransmissions [3], so as to retransmit lost packets before TCP, but this approach is inherently tied to TCP and not guaranteed to be the best one.

3 Simulation Setup

The performance results reported below are based on simulations with ns-2 [5], extended with additional error models, link layers and applications [6]. Each test was repeated 30 times with different random seeds. The results shown reflect average metric values from these 30 runs, as well as their 95% confidence intervals. The simulated topology is shown in Fig. 1: a Wired Server communicates with a Wireless Client via an Access Point. In all applications tested, the server was located at the wired end of the network and the client at the wireless end, hence the naming convention used. The wired link has a bandwidth of 10 Mbps and a propagation delay of 1 ms. Simulations using a 2 Mbps wired link with a propagation delay of 50 ms also support the conclusions reached in this paper.

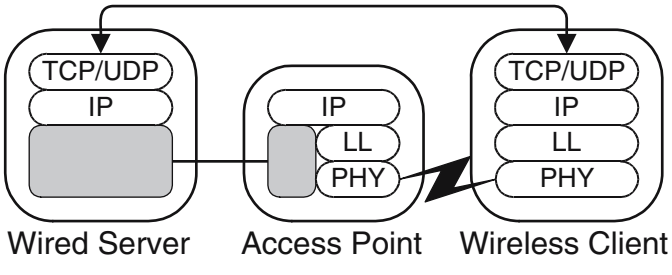


Fig. 1. Simulated network topology

The wireless link has a bandwidth of 64 Kbps, a propagation delay of 50 ms and uses a frame size of 250 bytes plus a header; these are typical characteristics for cellular links where bit interleaving inflates propagation delay. To avoid packet fragmentation, each application also uses 250 byte packets. Two error models were used for the wireless link. In the *Uniform* error model each frame may be independently lost with a probability of 1.5%, 2.5%, 5.4% or 9.8%. In the *Two State* error model the link can be either in a good state, with a bit error rate of 10^{-6}, or in a bad state, with a bit error rate of 10^{-2}. Both states have exponential durations, with the average duration of the good state being 10 s and the average duration of the bad state being 100 ms, 200 ms, 500 ms or 1000 ms. We have experimentally found that with these parameters the average *Frame Loss Rate* (FLR) of the Two State model is 1.5%, 2.5%, 5.4% or 9.8%, matching the FLRs used for the Uniform model. Note that the error processes in

each link direction were identical but independent. To establish a performance baseline, we also show results with no errors.

To evaluate the Selective Repeat variants presented below, we used Web Browsing, the most popular application on the Internet [7], over TCP Reno with 10 ms granularity timers. In Web Browsing a client accesses *pages* containing text, links to other pages and embedded objects, stored on a server. The client-server interaction consists of *transactions*: the client requests a page from a server, the server returns the page which contains pointers to embedded objects, the client requests each embedded object, and the server returns them, completing the transaction. The next transaction begins when the client requests another page. The ns-2 HTTP module provides empirical distributions for request, page and embedded object sizes, as well as for the number of objects per page [7]. Only one transaction was in progress at any time with no pauses between transactions. The performance metric used was Web Browsing throughput, defined as the amount of all *application* data transferred from the server to the client divided by time taken. Client requests only influence throughput indirectly, by introducing delays. All results shown reflect the state at the end of the last completed transaction during the simulated period, which was 2000 s.

Contention is provided by a UDP real-time Media Distribution application. This application approximates a lecture where a speaker sends audio, and possibly video, to an audience including a wireless client. The speaker alternates between *talking* and *silent* states with exponential durations, averaging 1 s and 1.35 s, respectively [8], transmitting media only in the talking state. Packets are transmitted isochronously at a rate of 56 Kbps, consuming 87.5% of the available bandwidth in the talking state, but only 37.5% of the available bandwidth on average. As a result, the bandwidth available for Web Browsing is abruptly modified whenever Media Distribution changes state. It should be noted that no retransmissions are performed for the, delay sensitive, Media Distribution application. To be more exact, the Media Distribution data stream bypasses the reliable link layer protocol used by the Web Browsing data stream.

4 Fixed Selective Repeat

In past research we have found the Selective Repeat protocol to offer excellent performance for TCP applications such as Web Browsing [2], without requiring TCP awareness. We have therefore decided to use it to study the interplay between link sharing and retransmission timers. In Selective Repeat, the sender transmits link layer frames in sequence within a transmission window of N frames, buffering them for possible retransmission. The receiver accepts frames within a reception window of N frames; if a frame is received in sequence it is delivered to the higher layer, the window slides upwards and an ACK is returned to the sender, confirming reception of all frames up to the one delivered. When the sender receives an ACK, it drops the buffered frames covered by it and also slides its window upwards.

When a frame arrives out of sequence at the receiver, it is buffered but not delivered, since the gap in the sequence indicates that some frames were lost; a *negative acknowledgment* (NACK) is returned for each missing frame to the sender, and the sender retransmits each NACKed frame. When missing frames arrive, the receiver delivers to the higher layer all frames that are now in sequence, slides its window upwards and returns an ACK covering all delivered frames. To reduce protocol overhead, in our implementation we delay returning an ACK for a short interval, trying to piggyback it into a data frame traveling in the reverse direction. If the interval expires, the ACK is sent as a separate frame. NACKs on the other hand are always sent immediately as separate frames.

If some ACKs and/or NACKs are lost, the sender may exhaust its transmission window, thus becoming unable to proceed. To prevent this, the sender starts a retransmission timer after sending each frame. If the timer expires before an ACK arrives for that frame, the frame is assumed lost and retransmitted. Many Selective Repeat variants exist, mostly differing on how NACKs are handled [9]. The variant used here allows each missing frame to be NACKed multiple times, a feature called *multireject*; we have also tested two simpler protocol variants, both of which support the conclusions reached in this paper.

We will now examine the performance of Web Browsing over Selective Repeat with fixed timers, in order to assess the effects of contention. Figure 2 shows the Web Browsing throughput achieved under the Uniform error model with a range of fixed timeout values from 0.9 s to 1.3 s, in 0.1 s increments. Throughput is maximized with the lowest timeout value, and it is progressively decreased as the timeouts are increased. The 9.8% performance gap, i.e. the difference between the best and worst options at a FER of 9.8%, is 13.4%. When contention is introduced however, the situation is completely reversed, as Fig. 3 shows: in this case lower timeout values perform worst, while higher timeouts lead to progressive improvement. In this case the 9.8% performance gap is 21.7%, but in the opposite direction than when no contention exists.

In the Two State error model, the situation without contention is not that clear. As Fig. 4 shows, it is hard to even distinguish between the various fixed timeout options, since the 9.8% performance gap is only 1.4%. However, when contention is introduced, Fig. 5 shows that throughput degrades as the timeouts are decreased, exactly as with the Uniform error model. Indeed, in this case the 9.8% performance gap is 40.9%, higher than with the Uniform error model.

These results indicate that under both wireless models, it is not possible to select a fixed retransmission timer value that will optimize overall Web Browsing performance: contention generally increases the optimal timeout value, due to the corresponding increase in the effective RTT. The results with contention show that a timeout value that provided excellent performance without contention, may exhibit terrible performance when contention is introduced. Therefore, with fixed timers the best we can do is to choose a timer that will provide a good compromise, rather than optimal performance.

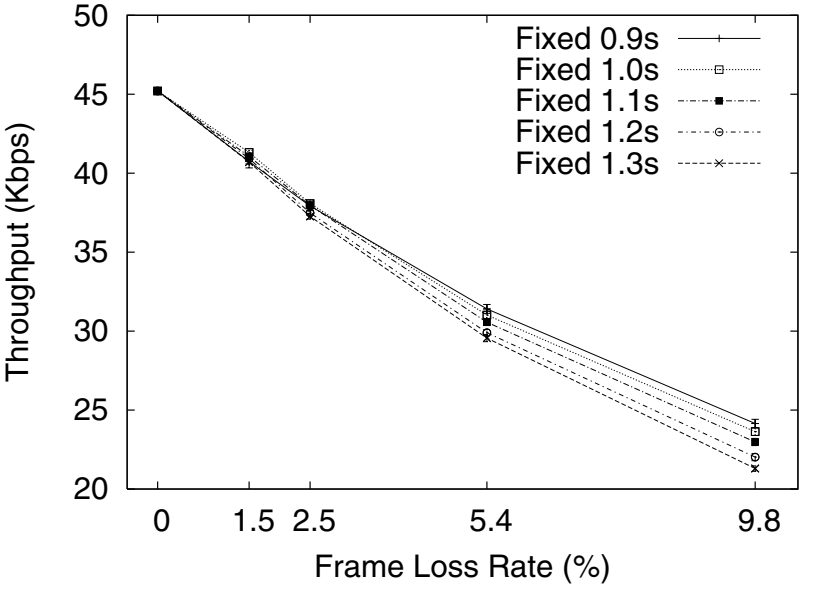


Fig. 2. Web Browsing Throughput without Contention (Uniform): Fixed Timeouts

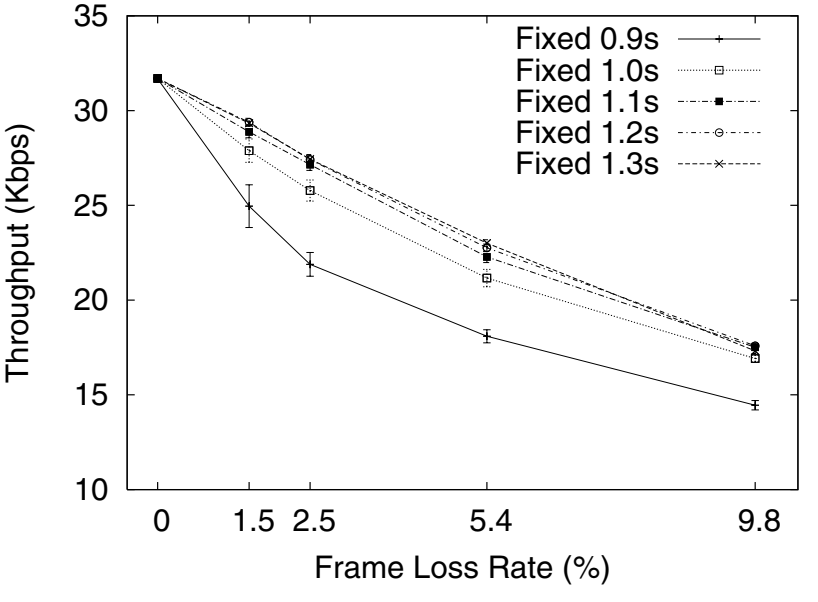


Fig. 3. Web Browsing Throughput with Contention (Uniform): Fixed Timeouts

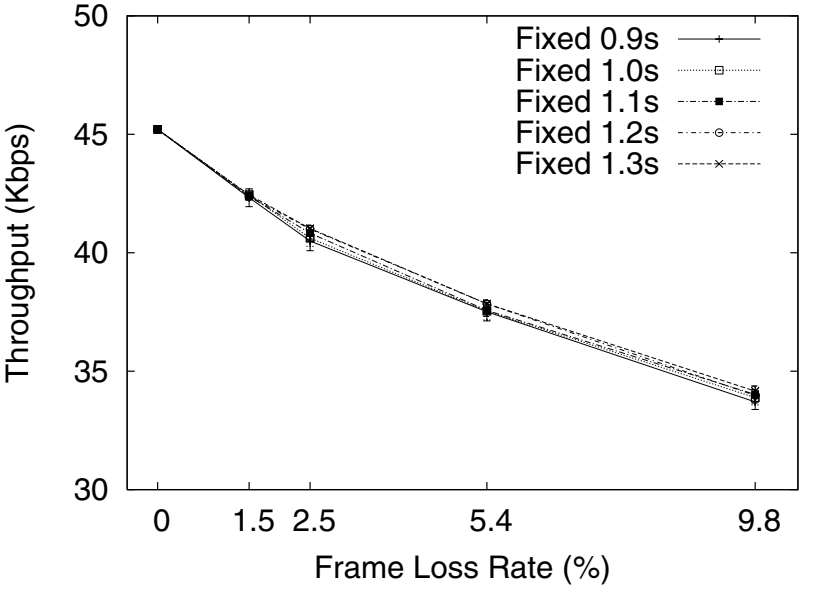


Fig. 4. Web Browsing Throughput without Contention (Two State): Fixed Timeouts

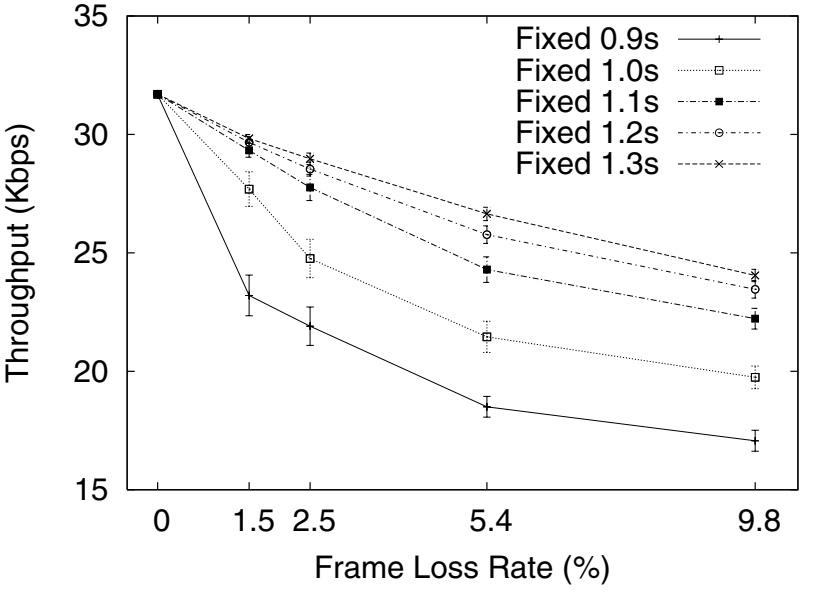


Fig. 5. Web Browsing Throughput with Contention (Two State): Fixed Timeouts

5 Adaptive Selective Repeat

As shown in Sect. 4, contention for the link has a dramatic effect on performance: when competing sessions start and stop, the available bandwidth changes, influencing the effective RTT. It is thus desirable for a reliable link layer protocol to appropriately adapt its retransmission timers. To this end, we modified Selective Repeat to track the RTT in a manner similar to TCP [10]. For every packet transmitted or retransmitted, the sender notes its transmission time. When an ACK arrives for the packet, the difference between the current time and the transmission time provides an RTT *sample*. We use these samples to update smoothed estimates for the RTT, *srtt*, and its variance, *srttvar*, as follows:

$$srtt = 0.875 * srtt + 0.125 * sample \ . \tag{1}$$

$$srttvar = 0.75 * srttvar + 0.25 * (sample - srtt) \ . \tag{2}$$

As the effective RTT fluctuates, the estimators (1) and (2) follow its progress in a smoothed manner: they react to changes with a time lag and are not dramatically affected by sporadic extreme values. Note that the smoothing factors used are the same as those used by TCP, therefore these calculations can be performed very efficiently using integer arithmetic [10]. After updating the estimators, we calculate the new value to be used for the retransmission timers, *rtxto*, as follows:

$$\text{Uniform error model}: \ rtxto = 3 * srtt + 2 * srttvar \ . \tag{3}$$

$$\text{Two State error model}: \ rtxto = 4 * srtt + 0 * srttvar \ . \tag{4}$$

Note that both (3) and (4) differ from the formula used by TCP, which is $rtxto = 1 * srtt + 4 * srttvar$. We have experimentally found that these formulas perform very well for the corresponding wireless error models [11], in contrast to the plain TCP formula, whose performance will be discussed below.

Our adaptive timeout scheme calculates samples from *every* packet acknowledged, with three exceptions, meant to avoid inaccurate samples. First, NACKs do not provide samples, since they do not reflect reception of the NACKed frame. Second, when an ACK covers multiple frames, only the last frame acknowledged provides a sample, since the previous ones may have been received long ago. Third, when duplicate ACKs arrive, only the first ACK provides a sample; the following ones are ignored, since they do not offer additional information.

We will now we examine the performance of Web Browsing over Selective Repeat with both adaptive and fixed timeouts. Figure 6 shows the Web Browsing throughput achieved under the Uniform error model with our *Adaptive* scheme, as well as with fixed timeout values of 0.9 s, 1.1 s and 1.3 s. We also show performance over the *Raw Link*, that is, without any link layer error recovery, and over *Plain TCP*, that is, when the standard TCP formula is used for timeout adaptation. Our adaptive scheme performs better than all fixed options: the 9.8% performance gap of our scheme over the fixed 1.1 s option, previously found to be a good compromise, is 17%, while plain TCP is worse than all fixed options.

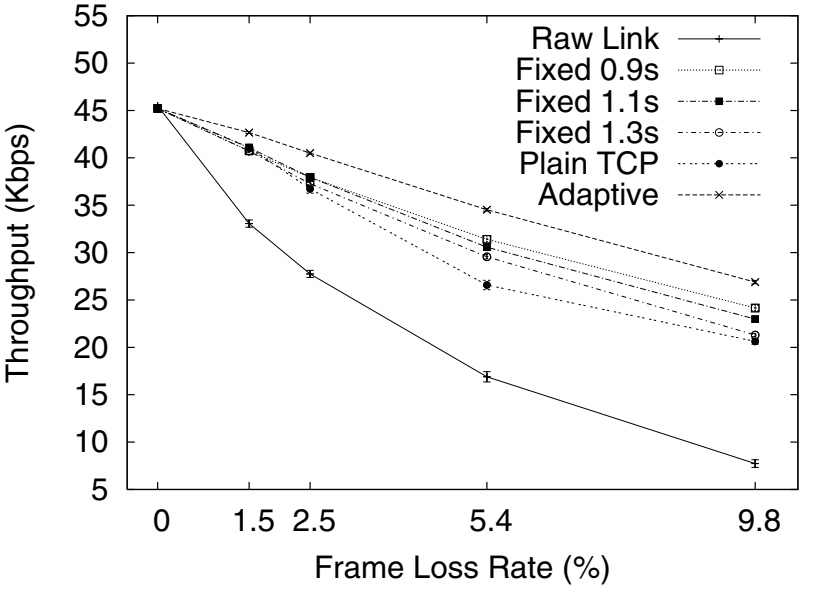


Fig. 6. Web Browsing Throughput without Contention (Uniform): Fixed vs. Adaptive Timeouts

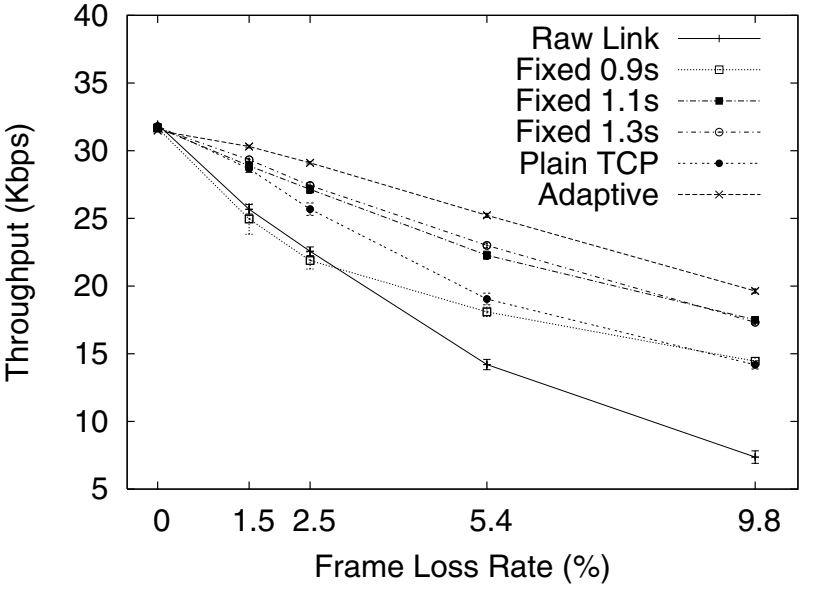


Fig. 7. Web Browsing Throughput with Contention (Uniform): Fixed vs. Adaptive Timeouts

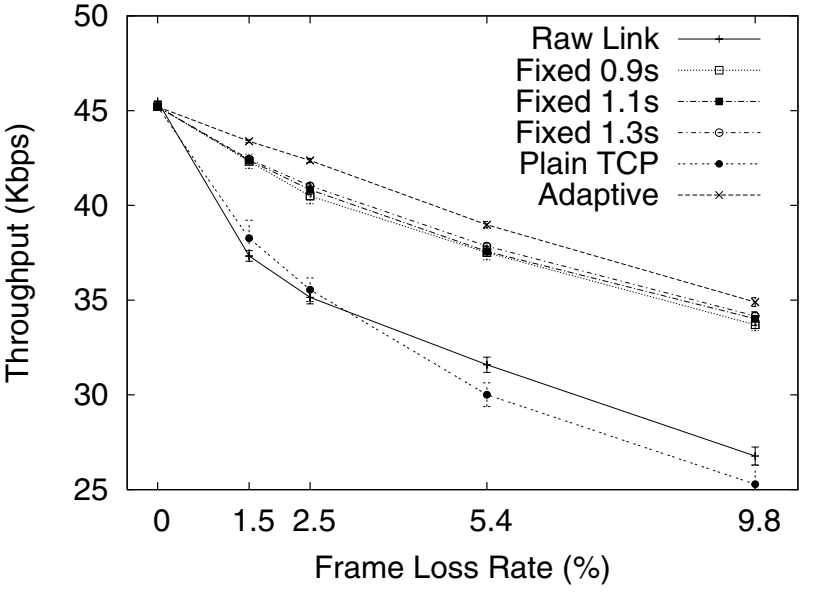


Fig. 8. Web Browsing Throughput without Contention (Two State): Fixed vs. Adaptive Timeouts

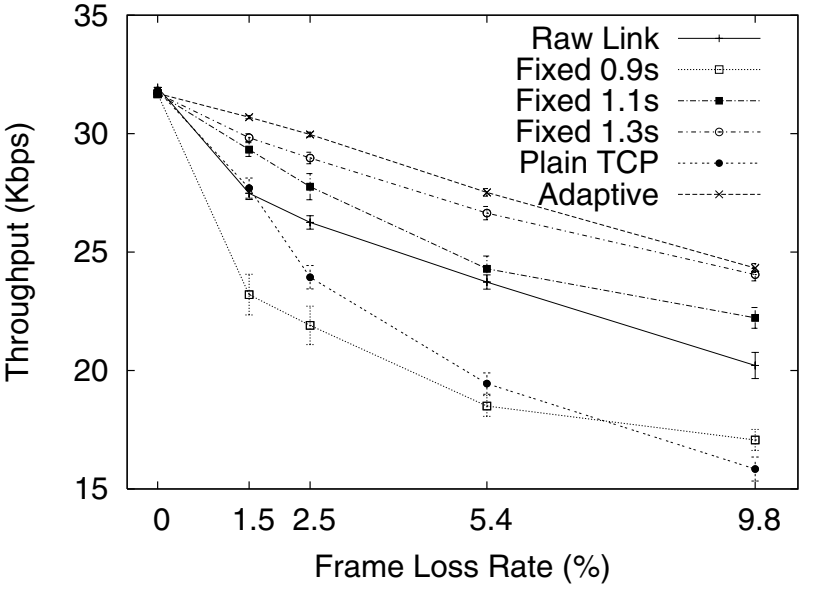


Fig. 9. Web Browsing Throughput with Contention (Two State): Fixed vs. Adaptive Timeouts

When contention is introduced, Fig. 7 shows that our adaptive scheme again performs best; in this case the 9.8% performance gap over the fixed 1.1 s option is 12.1%. Interestingly, with the lowest fixed timeout, i.e. 0.9 s, which was the best option without contention, performance with contention can be worse than without any error control, indicating that timers expire early, leading to redundant retransmissions. The plain TCP formula also suffers from the same problem, therefore it only manages to beat the worst fixed timeout option.

The results with the Two State error model are very similar. Figure 8 shows that without contention our adaptive scheme outperforms all fixed schemes, with a 9.8% performance gap over the fixed 1.1 s option of 2.6%. When contention is introduced, Fig. 9 shows that our adaptive scheme again performs best, with a 9.8% performance gap over the fixed 1.1 s option of 9.5%. In this case, the use of a fixed timeout of 0.9 s, which was quite acceptable without contention, is always worse than no error control at all. Furthermore, it is clear that the plain TCP formula is completely inappropriate for the Two State error model, as it nearly always leads to lower performance than without any error recovery.

These results indicate that with Web Browsing our adaptive timeout scheme performs much better than the fixed scheme with the compromise timeout of 1.1 s, and, indeed, better than any of the fixed timeout options tested. More importantly, our adaptive scheme performs excellent without manual tuning regardless of the level of contention, in contrast to the fixed schemes where the choice of an optimal timeout value requires awareness of the, generally unknown, level of congestion over the link. Furthermore, our results show that the equation used by TCP is far from optimal, and may even be worse than performing no error control at all. Therefore, simply using the TCP policy is not enough.

We conclude this section with a brief sensitivity analysis of the parameters used for the adaptive scheme under each wireless error model. In previous work we have explored the parameter space for the coefficients of *srtt* and *srttvar*, concluding that formulas (3) and (4) work best for the Uniform and Two State error models, respectively [11]. We have also compared our basic scheme, referred to as adaptive standard, against two variations: in the adaptive fast case, equations (1) and (2) are modified to $srtt = 0.75 * srtt + 0.25 * sample$ and $srttvar = 0.5 * srttvar + 0.5 * (sample - srtt)$, respectively, i.e. the estimators adapt faster to prevailing conditions; conversely, in the adaptive slow case, equations (1) and (2) are modified to $srtt = 0.9375 * srtt + 0.0625 * sample$ and $srttvar = 0.875 * srttvar + 0.125 * (sample - srtt)$, respectively, i.e. the estimators adapt slower. We omit these results for brevity, as all three variants of our adaptive scheme have only marginal performance differences, indicating that the scheme is relatively insensitive to the exact choice of the filter coefficients used.

6 Conclusions and Future Work

We have discussed the problems faced by reliable link layer protocols when sharing a wireless link with competing link layer sessions, using Selective Repeat as an example. Our measurements of Web Browsing performance indicate that the

optimal fixed retransmission timer values strongly depend on the level of contention for the link. We therefore proposed an adaptive Selective Repeat variant that dynamically sets its retransmission timers based on prevailing conditions, using a scheme similar to TCP. Our measurements indicate that this adaptive scheme outperforms its fixed counterparts regardless of the level of contention and frame loss rate, under two very different wireless error models. We have also found that, while our adaptive scheme is relatively insensitive to the filter coefficients used to smooth the average RTT and RTT variance estimators involved in the adaptive calculation of the timeout values, the actual coefficients used by TCP for this calculation are suboptimal for our link layer environment.

Our adaptive Selective Repeat approach is not the only way to overcome the issues introduced by contention at the link layer. The *Radio Link Control* (RLC) protocol used in UMTS networks in its Acknowledged Mode [12] does not use retransmission timeouts at the sender, relying solely on status information from the receiver to trigger retransmissions. Since both ACKs and NACKs may be lost however, either the RLC sender or the RLC receiver must periodically probe for or return status reports, respectively. We are currently implementing the UMTS RLC protocol in our simulator with the aim of comparing its performance against our adaptive Selective Repeat approach.

References

1. Xylomenos, G., Polyzos, G.C.: Internet protocol performance over networks with wireless links. IEEE Network **13** (1999) 55–63
2. Xylomenos, G., Polyzos, G.C.: A multi-service link layer architecture for the wireless Internet. International Journal of Communication Systems **17** (2004) 553–574
3. Balakrishnan, H., Padmanabhan, V.N., Seshan, S., Katz, R.H.: A comparison of mechanisms for improving TCP performance over wireless links. In: Proc. of the ACM SIGCOMM '96. (1996) 256–267
4. Kent, S., Atkinson, R.: IP encapsulating security payload (ESP). RFC 2406 (1998)
5. UCB/LBNL/VINT: Network Simulator - ns (version 2). (Available at http://www.isi.edu/nsnam)
6. Xylomenos, G.: Multi service link layers for ns-2. (Available at http://www.mm.aueb.gr/~xgeorge/codes/codephen.htm)
7. Mah, B.A.: An empirical model of HTTP network traffic. In: Proc. of the IEEE INFOCOM '97. (1997) 592–600
8. Nanda, S., Goodman, D.J., Timor, U.: Performance of PRMA: a packet voice protocol for cellular systems. IEEE Trans. on Vehicular Technology **40** (1991) 584–598
9. Brady, P.T.: Evaluation of multireject, selective reject, and other protocol enhancements. IEEE Trans. on Communications **35** (1987) 659–666
10. Jacobson, V.: Congestion avoidance and control. In: Proc. of the ACM SIGCOMM '88. (1998) 314–329
11. Xylomenos, G., Tsilopoulos, C.: Adaptive timeout policies for wireless links. In: Proc. of the International Conference on Advanced Information Networking and Applications. Volume 1. (2006) 497–502
12. 3rd Generation Partnership Project (3GPP): Radio Link Control (RLC) protocol specification (Release 6). Technical Specification 25.322, V6.2.0 (2004)

Performance Analysis of IEEE 802.11e WLANs with Hidden Stations and Heterogeneous Traffic

Mamun I. Abu-Tair and Geyong Min

Department of Computing, School of Informatics, University of Bradford,
Bradford, BD7 1DP, U.K
{m.i.a.abu-tair, g.min}@brad.ac.uk

Abstract. IEEE 802.11e Medium Access Control (MAC) mechanism has been recently proposed for supporting differentiated Quality-of-Services (QoS) in Wireless Local Area Networks (WLANs). Heterogeneous traffic generated by wireless multimedia applications and hidden stations arisen from the wireless transmission power constraints have significant impact on the performance of MAC protocols. This study performs extensive simulation experiments and conducts comprehensive performance evaluation of the IEEE 802.11e Enhanced Distributed Channel Access (EDCA) protocol in WLANs with hidden stations and heterogeneous traffic. For this purpose, non-bursty Poisson, bursty ON/OFF, and fractal-like self-similar processes with high variability are used to model and generate heterogeneous network traffic. The performance results have shown that the protocol is able to achieve differentiated throughput, access delay and medium utilization. However, the hidden stations can degrade the throughput and medium utilization and also increase the medium access delay greatly in the presence of heterogeneous traffic.

1 Introduction

Wireless Local Area Networks (WLANs) are being deployed in enterprises all over the world and are becoming the fastest growing segment of the communication market due to its simplicity, flexibility and accessibility independent of location as well as its ability for wireless stations to roam throughout the business organizations. Current research and market analysis have shown that the worldwide shipments of WLAN units will keep growing at an annual rate of 42% through 2007 [23]. In order for WLANs to be widely accepted and to support ever-growing wireless and mobile applications, the Institute of Electrical and Electronics Engineers (IEEE) have ratified 802.11 standards to ensure the compatibility and reliability among the network devises. IEEE 802.11 standards specify a common Medium Access Control (MAC) protocol which is responsible for managing and maintaining communication between wireless network devices and coordinating access to a shared channel.

The dominating mechanism of the IEEE 802.11 MAC is called the Distributed Coordination Function (DCF), which relies on Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) algorithm to access the shared medium. The main objective of CSMA/CA is to avoid stations transmitting packets at the same time, which can lead to collisions and corresponding retransmissions [4, 10, 12, 21].

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 171 – 182, 2006.
© Springer-Verlag Berlin Heidelberg 2006

However, hidden stations in WLANs often cause serious problems and performance degradation. A pair of stations are said to be hidden from each other if the transmission from one station cannot be heard by the other. As a result, two or more hidden stations may transmit data at the same time, causing a collision and a MAC failure. Thus, additional collision avoidance protocol becomes necessary to combat the hidden station problem. Apart from the common CSMA/CA techniques, the DCF further reduces the possibility of collisions using the popular collision avoidance scheme that consists of channel reservation frames (i.e., Request-To-Send and Clear-To-Send).

Due to its contention-based mechanism, the legacy IEEE 802.11 DCF cannot support differentiated Quality-of-Service (QoS) requirements of multimedia applications. As a result, the IEEE 802.11 working group has recently standardized an extended version (IEEE 802.11e) that defines two mechanisms for the support of QoS differentiation: Enhanced Distributed Channel Access (EDCA) and Hybrid Coordination Function (HCF) Controlled Channel Access (HCCA) [22]. EDCA delivers traffic based on differentiated Access Categories (ACs), which can be achieved by varying the amount of time for which a station senses the channel to be idle before back-off, or the length of the contention window, or the duration in which a station may transmit after occupying the channel [22]. The stations with lower-priority traffic must wait longer than those with high-priority before accessing the medium [22].

Many recent studies [8, 12, 13, 19] have conducted performance analysis and evaluation of IEEE 802.11e MAC protocols using simulation experiments. For instance, [12] has investigated and compared different MAC mechanisms for supporting QoS in WLANs including Point Coordination Function (PCF), Distributed Fair Scheduling (DFS), Blackburst and EDCA. Their results have showed that the best performance is achieved by Blackburst and have demonstrated that PCF and EDCA are able to support good service differentiation. [8] has carried out performance evaluation of IEEE 802.11e under different types of traffic (such as VoIP, video) with the limitation that each station generates only one single class of traffic. [13] has evaluated the performance of IEEE 802.11e in the single and overlapping Access Point (AP) environments using traffic models with negative-exponentially distributed inter-arrival times. This study has focused on the analysis of the effectiveness and limitations of IEEE 802.11e in such environments. [19] has evaluated the effectiveness of various IEEE 802.11e QoS mechanisms for supporting voice, video and data applications separately in individual scenarios.

Many measurement studies [3, 6] have shown that traffic generated by multimedia applications exhibits heterogeneous properties. For example, [6] has adopted the well-known bursty On-Off model to capture voice traffic where the talkspurt is modeled by ON stage and the silence period is modeled by OFF stage with exponential ON and OFF periods. More recently, [3] based on high-quality measurement traces has indicated that the compressed Variable-Bit-Rate (VBR) video exhibits fractal-like self-similar and Long Range Dependent (LRD) properties (i.e., traffic burstiness and correlations appearing over many time scales). Moreover, such self-similar traffic behaviors have been found in WLANs [18]. The study [2] has recently reported performance results of EDCA in the presence of heterogeneous multimedia traffic.

However, [2] did not take hidden stations into account, which have great effects on the performance of MAC protocols and communication networks. To fill this gap, this study performs extensive simulation experiments and investigates the throughput, access delay and medium utilization of the IEEE 802.11e EDCA protocol in WLANs with hidden stations and heterogeneous non-bursty Poisson, bursty ON/OFF, and fractal-like self-similar traffic. The performance results have demonstrated that the protocol is able to achieve satisfying QoS differentiation for heterogeneous multimedia applications. However, this protocol suffers from the low medium utilization due to the overhead generated by transmission collisions and back-off processes. Furthermore, this study has shown that the hidden stations can degrade the throughput and medium utilization and also increase the medium access delay greatly in the presence of heterogeneous traffic.

The rest of the paper is organized as follows: Section 2 starts with the introduction to MAC mechanism and then reviews the problem of hidden stations. Section 3 describes the simulation scenarios and the setting of simulation parameters. Section 4 presents how to generate heterogeneous traffic. Section 5 presents and analyses the performance results obtained from our simulation experiments. Finally, Section 6 concludes this study.

2 Wireless Local Area Networks (WLANs)

Recently IEEE 802.11 WLANs have gained a prevailing position in the business enterprises. The IEEE 802.11 Medium Access Control (MAC) layer specifies the functions and protocols required for control and access the wireless medium.

2.1 Medium Access Control (MAC)

The IEEE 802.11 MAC protocol offers two different forms to support shared access to wireless channels: a DCF and an optional PCF [10]. The former, dominating MAC mechanism implemented in the IEEE 802.11-compliant products, is a contention-based MAC protocol where each station decides the time to start transmission using the CSMA/CA mechanism [10, 21]. Moreover, an additional random binary exponential time called back-off time is adopted to reduce the probability of collisions. The back-off time is chosen randomly from the interval $[0, cw]$, where cw represents the contention window [21]. The stations start down-counting its back-off counter by one as long as the medium has been detected idle for at least the minimum duration called DCF Inter-Frame Space (DIFS). If the medium gets busy due to other transmissions, the back-off counter pauses down-counting and resumes when the medium has been sensed idle for DIFS again [10, 21]. Transmission may proceed when back-off counter has reached zero. Upon detection of a collision, i.e., when the back-off counter of two or more stations reaches zero at the same time, the contention window is doubled [12]. When the destination station receives frame successfully, it sends an acknowledgment (ACK) frame back to the source station after a Short Inter-Frame Space (SIFS) duration. Additionally, to alleviate the hidden station problem, DCF uses optional Request-to-Send/Clear-to-Send (RTS/CTS) frames before packet transmission [21]. A station initiates the transmission process by sending an RTS frame. The destination station replies to the RTS frame with a CTS frame to confirm

the reservation of the shared medium. After that, the source station transmits a data packet to the destination station.

With the ever increasing popularity of WLANs, the support of QoS has become a critical issue on the success of IEEE 802.11 MAC protocols for future wireless communication. It is important to develop new medium access schemes that can support real-time multimedia applications with differentiated QoS requirements over WLANs. To overcome the drawback of the traditional DCF, EDCA has been proposed to provide differentiated and distributed channel accesses for four different Access Categories (ACs). Each AC has its own transmission queue and is labeled according to its application, i.e. AC_VO (voice), AC_VI (video), AC_BE (best-effort), and AC_BK (background). As illustrated in Figure 1, individual AC contends to access the medium and is differentiated by different Inter-Frame Space (AIFS) values, Transmission Opportunity (TXOP) limits and minimal/maximal contention windows ($cw_{\min}$, $cw_{\max}$) [13, 14]. Choosing a small contention window causes the station to gain priority over others with the larger contention window. After any unsuccessful transmission, the contention window increases by multiplying the old contention window with the persistence factor. The persistence factor determines the degree of increase in the contention window in the event of collision. Higher priority AC has smaller persistence factor.

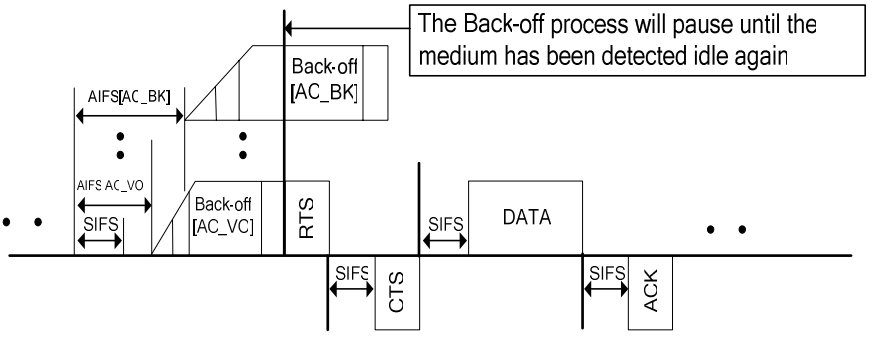


Fig. 1. RTS/CTS of IEEE 802.11e EDCA

2.2 Hidden Stations

Due to the constraints of wireless transmission power, hidden stations often appear in WLANs where a receiver node lies within the transmission range of two other nodes which are mutually hidden, i.e., which cannot sense the transmission of each other. As illustrated in Figure 2, two wireless stations can sense the Access Point (AP) but cannot sense each other. Unfortunately, the phenomenon of hidden stations can degrade network performance substantially. The major effect of hidden stations is to cause collision at any stage of the transmission-receive process. As a result, both stations need to retransmit their data packets and start an extra back-off time which leads to the increase of delay. The retransmission overhead also affects the channel utilization and throughput of WLANs. To overcome this problem, IEEE 802.11 MAC

adopts RTS/CTS acknowledgment and hand-shaking mechanism. All stations receiving either RTS and/or CTS frames will defer down-counting its back-off counter for the specific duration. Therefore, collision may be avoided even though some stations are hidden from others. It is worth noting that the gain of network performance using RTS/CTS introduces the additional overhead of exchanging RTS/CTS frames. Thus, it is necessary and critical to investigate the performance of this scheme in the presence of hidden stations.

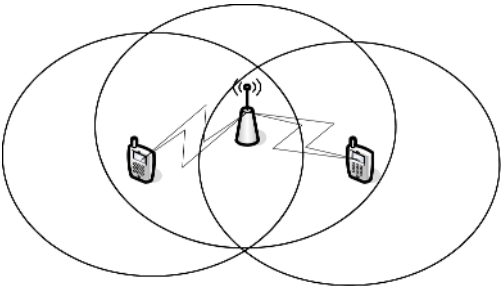


Fig. 2. Hidden station problem

3 Simulation Scenarios

The simulation scenarios studied by this research have been designed to investigate the performance of EDCA in hot spot areas of WLANs using well-known network simulator NS-2 and its wireless extensions developed by TKN [20].

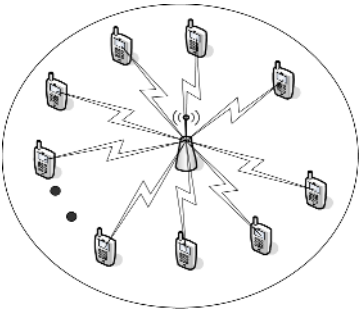


Fig. 3. AP with single set of mobile stations

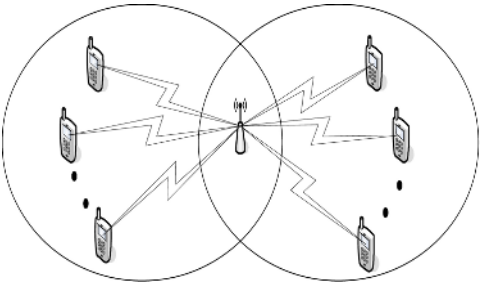


Fig. 4. AP with two sets of mobile stations

In order to carry out a comprehensive performance study of EDCA, two different scenarios are considered. As shown in Figure 3, the first scenario is composed of up to 20 mobile stations and an access point serving as a traffic sink. All stations are located within an Independent Basic Service Set (IBSS) such that every station is able to detect a transmission from others. Figure 4 illustrates the second scenario where there are two sets of mobile stations with up to 10 stations in each set. All stations of

the same set are able to detect transmission from other set members but cannot detect the transmission from the stations belonging to the other set, i.e., the stations in one set are hidden from all stations belonging to the other set. Each wireless station in both scenarios operates at a data rate of 24Mbps [21]. Table 1 lists the setting of physical (PHY) layer parameters which have been widely used by other performance studies on WLANs [13, 14, 20].

Table 1. PHY parameters for simulation scenarios

SlotTime	9 μs
CCATime	3 μs
RxTxTurnaroundTime	2 μs
SIFSTime	16 μs
PreambleLength	96 bits
PLCPHeaderLength	40 bits
PLCPDataRate	6 Mbps

4 Heterogeneous Traffic Models

Appropriate models that can accurately capture the properties of real-world network traffic are required for effective and reliable performance evaluation of the EDCA protocol. Due to the different types of WLAN services, heterogeneous traffic models should be used to capture the characteristics of various wireless applications. In our study, the lowest priority traffic (AC_BK) is modeled by the conventional Poisson arrival process with which the inter-arrival times of packets are exponentially distributed. Each station generates background traffic by sending packets with rate of 160 Kbit/s and size of 200 bytes; these parameters have been widely used in the previous network performance studies [13, 14].

However, the Poisson model cannot capture the bursty nature of network traffic. The highest priority traffic (AC_VO) generated by voice sources is captured by the well-known bursty ON/OFF model where the talkspurt is modeled by ON stage and silence period by OFF stage. In general, the talkspurt and silence durations of voice ON/OFF process are assumed to follow an exponential distribution with mean length of 1s and 1.35s [6], respectively. During the talkspurts period each source generates packets with size of 80 bytes, corresponding to a constant sending rate of 64 kbit/s. As there might be multiple voice sources, we assume there are five voice sources at each station.

Extensive measurement studies on high-speed video traffic have reached to the conclusions that such traffic possesses the self-similar property [3, 5]. Self-similar traffic is characterized by the phenomenon that the pattern of packet arrivals appears similar to itself when viewed over a wide range of time scales. Traffic generated by streaming applications such as real-time video (AC_VI) and best-effort video

(AC_BE) are modelled by self-similar processes in this study. According to the measurement results of Star Wars video trace [3, 5], each station generates real-time and best-effort video traffic with rate of 360 Kbit/s and Hurst parameter of 0.7. Hurst parameter is used to characterize the degree of traffic self-similarity. Such traffic can be generated by the superposition of many ON/OFF sources in which the ON and OFF periods follow Pareto distributions (i.e., with high variability or infinite variance) [15]. The lengths of the ON and OFF periods are determined by $K(x^{-1/a}-1)$ where x is a uniformally distributed random number between 0 and 1, $a=3-2H$, $K=m(a-1)$ and m is the mean length of ON and OFF period, respectively [11]. In the simulation experiments, we used the superposition of five Pareto-distributed ON/OFF flows with the mean ON period of 10ms and OFF period of 100ms to generate video traffic.

To support differentiated QoS, EDCA protocol parameters for the four different traffic ACs including $cw_{\min}$, $cw_{\max}$, $AIFS$ are selected following the IEEE 802.11e standard and are listed in Table 2.

Table 2. EDCA protocol parameters for different traffic categories

Parameters / AC	$cw_{\min}$	$cw_{\max}$	AIFS
AC_VO	$(cw_{\min}+1)/4-1=3$	$(cw_{\min}+1)/2-1=7$	2
AC_VI	$(cw_{\min}+1)/2-1=7$	$cw_{\min}=15$	2
AC_BE	$cw_{\min}=15$	$cw_{\max}=1023$	3
AC_BK	$cw_{\min}=15$	$cw_{\max}=1023$	7

5 Analysis and Evaluation of Various Performance Metrics

In this section we report the impact of hidden stations on the prioritisation capability and performance of IEEE 802.11e EDCA MAC protocol in terms of medium utilization, access delay and throughput in the presence of heterogeneous traffic.

5.1 Throughput

The average throughput is calculated as the mean volume of data that is actually delivered to the destination within each time unit. As an effort to show the effects of the hidden station on the performance of EDCA, Figure 5 compares the throughput of four traffic categories versus the number of active stations under two different scenarios: (1) single set scenario without hidden stations (illustrated in Figure 3) and (2) double set scenario with hidden stations (Figure 4). We can see from Figure 5 that the throughput in both scenarios increases as the traffic loads rise until there are 6 active stations. Beyond this point, the background traffic begins to decrease significantly in both scenarios due to its lowest priority. It is clear that the degradation

of throughput with hidden stations is faster. Moreover, when there are 8 active stations, the throughput of background traffic in the single set scenario is 0.64 Mbit/s and in the hidden stations scenario is 0.28 Mbit/s. This difference is due to the increasing probability of collisions caused by the hidden stations. Furthermore, the best-effort video traffic cannot carry on its load where the number of active stations is more than 6 in the hidden stations scenario and more than 8 in the single set scenario. Figure 5 also shows that the throughput of the real-time video streams starts to decrease when there are more than 8 stations in both scenarios. However, the highest priority voice traffic at the single set scenario keeps increasing until there are 12 active stations while at the hidden stations scenario the throughput starts to decrease immediately after there are 10 active stations. The figure shows clearly the deteriorated effect of hidden stations on the throughput of voice and real-time video traffic. For example, when there are 20 active stations the throughput of voice traffic is around 1.10 Mbit/s (for single set scenario) and 0.94 Mbit/s (for hidden stations scenario) and the throughput of real-time video traffic is around 0.86 Mbit/s and 0.53 Mbit/s, respectively.

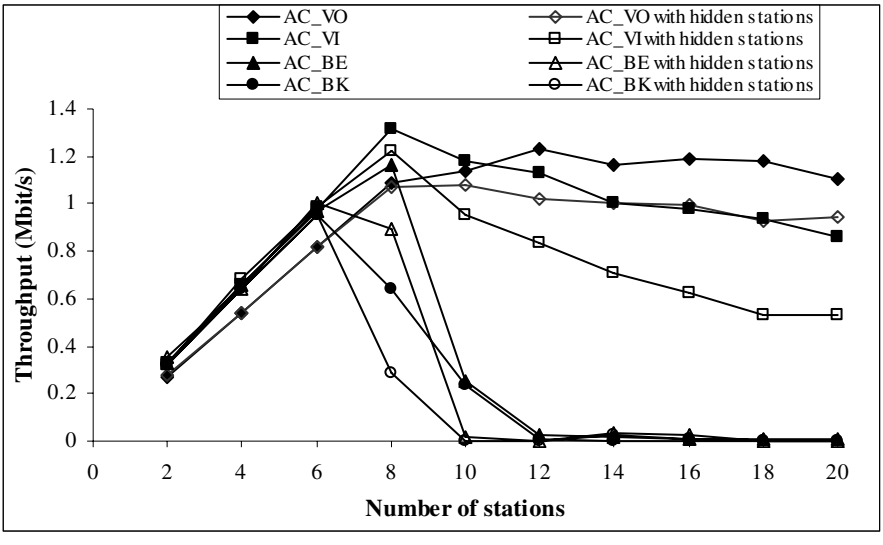


Fig. 5. Throughput of four traffic categories under two different scenarios

5.2 Access Delay

We measure the access delay to find out how well the EDCA accommodates real-time traffic, especially the voice and real-time video traffic. The access delay is defined as the time elapsed between the packet arrival from the higher layer to the MAC layer and the successful transmission on wireless medium. Figure 6 presents a comparison of the mean access delay with and without the existence of hidden stations, respictively. The figure reveals that both background traffic and best-effort video traffic suffer from the high access delay in both scenarios because of their lower priorities. When the number of active stations is beyond 6, background traffic and best-effort video traffic have the rare opportunity to capture the wireless medium. As

a result, the delay is going to become infinite when the number of active stations is more than 8. However, a close-up check can find that the mean access delay of the background and best-effort video traffic at the single set scenario is smaller than that at the hidden stations scenario. More specifically, when there are 8 active stations, the mean access delay for the best-effort video traffic is around 0.09 second (for the single set scenario) and 0.31 second (for the hidden stations scenario), respectively. In addition, the mean access delay of the real-time video and voice streams start to increase significantly when the network supports more than 8 active stations in both scenarios. The mean access delay of the real-time video traffic when there are 8 active stations is around 0.02 second (for the single set scenario) and 0.04 second (for the hidden stations scenario). Additionally, the mean delay of real-time video traffic reaches to 0.23 second (for the single set scenario) and 0.46 second (for the hidden stations scenario), respectively, and keeps increasing in a dramatic way when the number of active stations is 10 in both scenarios. Furthermore, the mean access delay of the real-time video traffic at the single set scenario is always less than that at the hidden stations scenario for all cases. It is also interesting to see that the delay experienced by the voice and real-time video traffic is much different because the latter has to wait for longer than the former until taking the opportunity to utilize the wireless medium. Moreover, Figure 6 shows that voice traffic has nearly the same mean access delay in both the single set scenario and hidden stations scenario.

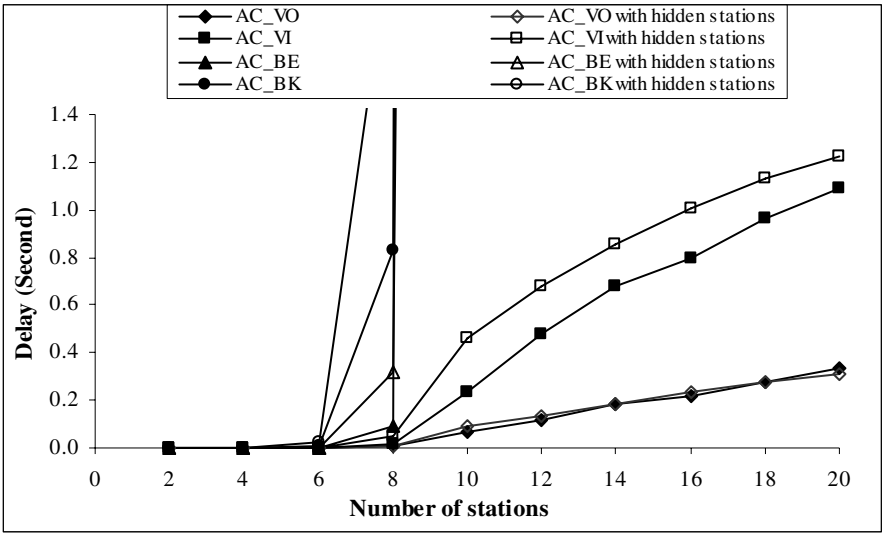


Fig. 6. Mean access delay of four traffic categories under two different scenarios

5.3 Medium Utilization

Medium utilization is referred to as the percentage of time that is used for successful transmission. Figure 7 depicts the medium utilization of four traffic categories under two different scenarios (with and without hidden stations, respectively). We can observe that all types of traffic categories have higher medium utilization at the single

set scenario than the hidden stations scenario. Furthermore it can be seen that the medium utilization of background and best-effort video traffic keep increasing as the load increases until there are 6 active stations in single set scenario and 8 stations in the hidden station scenario. After these points they cannot obtain increasing medium utilization because of giving the chance for the higher traffic categories to occupy the medium. Additionally, Figure 7 shows that the hidden stations have a deteriorated impact on the background and best-effort video traffic. It is also clear that real-time video traffic obtains higher medium utilization than background and best-effort video traffic and its medium utilization keeps growing up until there are 8 active stations. Beyond this point it starts to lose some medium utilization in both scenarios. Figure 7 also reveals that the voice traffic obtains the highest medium utilization in the both scenarios due to its highest priority. In addition, the figure shows clearly the effect of the hidden stations on the voice and real-time video traffic. Specifically, when there are 18 active stations the medium utilization for real-time video is around 6.5% (for the single set scenario) and 3.9% (for the hidden stations scenario). It is worth noting that the maximum total medium utilization of the WLAN is very low (around 36% (for the single set scenario) and 27% (for the hidden stations scenario)). This is because much capacity is wasted during the process of packet collisions and back-off. The increasing probability of packet collisions under heavy traffic and hidden stations can significantly degrade the medium utilization of the system.

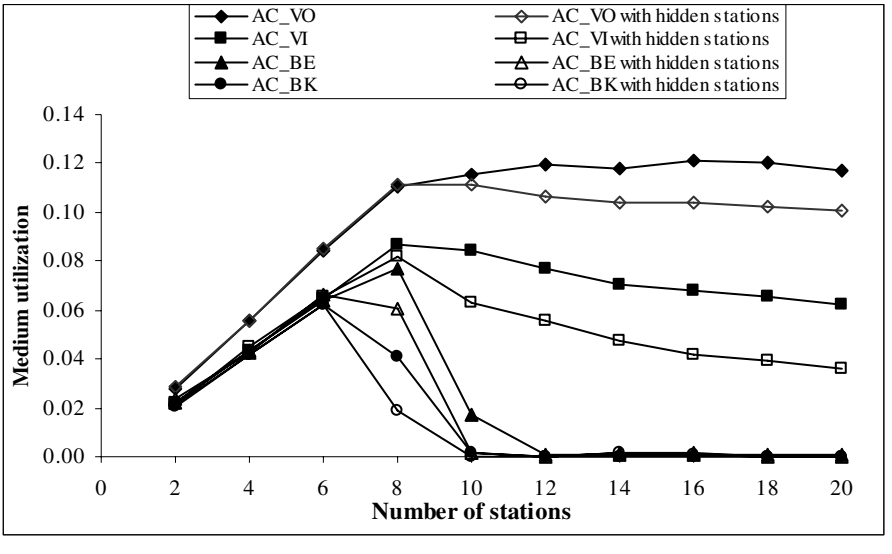


Fig. 7. Medium utilization of four traffic categories under two different scenarios

6 Conclusions

Wireless Local Area Networks (WLANs) offer flexible data communication systems to provide location independent network access between computation and communication devices using waves rather than a cable infrastructure. Many MAC

protocols have been proposed to mange and control the shared wireless medium. Among such protocols, the IEEE 802.11e EDCA mechanism aims to enhance the traditional 802.11 DCF protocols with differentiated QoS provisioning. Performance studies on the EDCA protocol have been widely conducted and reported in the literature. Different from most existing work, this study has used the well-known network simulator NS-2 to evaluate the performance of IEEE 802.11e MAC protocol in WLANs with hidden stations and in the presence of heterogeneous non-bursty Poisson, bursty ON/OFF, and self-similar traffic generated by wireless multimedia applications traffic. Performance results have shown that this protocol can support differentiated throughput, access delay and medium utilization among various access categories with or without the existence of hidden stations. However, the results have also demonstrated that IEEE 802.11e EDCA performs better without hidden stations. On the other hand, the simulation experiments have showed that IEEE 802.11e EDCA suffers from the low medium utilization due to the overhead generated by transmission collisions and back-off processes.

References

1. Aad, I. and Castelluccia, C.: Differentiation mechanisms for IEEE 802.11. *Proc.20th Annual Joint Conference of the IEEE Computer and Communications Societies. IEEE (INFOCOM'2001).* vol. 1. (2001) 209-218
2. Abu-Tair, M.I. and Min, G.: Performance evaluation of an enhanced distributed channel access protocol under heterogeneous traffic. *Proc. 5th Int. Workshop on Performance Modeling, Evaluation, and Optimization of Parallel and Distributed Systems (PMEO-PDS '2006 in conjunction with IPDPS '2006).* IEEE Computer Society Press. Rhodes Island. Greece. April 25-29. (2006). CD-ROM.
3. Beran, J., Sherman, R., Taqqu, M.S. and Willinger, W.: Long-range dependence in variable-bit-rate video traffic. *IEEE Trans. Communications.* vol. 43. (1995) 1566-1579
4. Choi, S., Prado, J.D., Shankar, S. and Mangold, S.: IEEE 802.11e Contention-based channel access (EDCF) performance evaluation. *Proc. IEEE Information and Communication Conference (ICC'2003).* vol. 2. (2003) 1151-1156
5. Chen, B., Lucic, Z. and Trajkovic, L.: Simulation and wavelet analysis of packet traffic. http://www.ensc.sfu.ca/people/faculty/ljilja/cnl/pdf/bruce_asi2002.pdf. (2005)
6. Cox, R.: Three new speech coder from the ITU cover a range of applications. *IEEE Communication Magazine.* vol. 35. no. 9. (1997) 40-47
7. Gao, D., cai, J., Bao, P. and He, Z.: MPEG-4 video streaming quality evaluation in IEEE 802.11e WLANs. *Proc. IEEE International Conference on Image Processing (ICIP).* vol. 1. (2005) 197-200
8. Grilo, A. and Nunes, M.: Performance evaluation of IEEE 802.11e. *Proc. 13th IEEE International Symposium, Personal, Indoor and Mobile Radio Communications (PIMRC'2002).* vol. 1. (2002) 511-517
9. Garrett, M. and Willinger, W.: Analysis, modelling and generation of self-similar VBR video traffic. *Proc. Conference on Communications Architectures, Protocols and Applications, ACM Special Interest Group on Data Communication (SIGCOMM'1994).* (1994) 269-280
10. Gast, M.: 802.11 wireless networks: the definitive guide. vol.1. O'Reilly. (2002)
11. Hassan, M. and Jain, R.: High performance TCP/IP networking concepts, issues and solutions. Pearson Prentice Hall. (2004)

12. Lindgren, A., Almquist, A. and Schelen, O.: Evaluation of quality of service schemes for IEEE 802.11 wireless LANs. *Proc. 26th Annual IEEE Conference on Local Computer Networks (LCN'2001).* (2001) 348 – 351
13. Mangold, S., Choi, S., Hiertz, G., Klein, O. and Walke, B.: Analysis of IEEE 802.11e for QoS support in wireless LANs. *IEEE Wireless Communications.* vol. 10. no. 6. (2003) 40–50
14. Mangold, S., Choi, S., May, P., Klein, O., Hiertz, G. and Stibor, L.: IEEE 802.11e wireless LAN for quality of service, *Proc. European Wireless Conference.* Italy. (2002)
15. Paxson, V. and Floyd, S.: Wide-area traffic: the failure of poisson modelling. *IEEE/ACM transaction on networking.* vol. 3. no. 3. (1995) 226-244
16. Qiang, N.: Performance analysis and enhancements for IEEE 802.11e wireless networks. *IEEE Mag. Network.* vol. 19. no. 4. (2005) 21-27
17. Tao, Z. and Panwar, S.: Throughput and delay analysis for the IEEE 802.11e enhanced distributed channel access. *IEEE Trans. Communications.* vol. 54. no. 4. (2006) 596-603
18. Tickoo, O. and Sikdar, B.: On the impact of IEEE 802.11 MAC on traffic characteristics. *IEEE Journal on Selected Areas in Communications.* vol. 21. no. 2. (2003) 189-203
19. Truong, H.L. and Vannuccini, G.: Performance evaluation of the QoS enhanced IEEE 802.11e MAC layer. *Proc. 57th IEEE Semi-annual, Vehicular Technology Conference (VTC'2003).* vol.2. (2003) 940-944
20. Wietholter, S. and Hoene, C.: Design and verification of an IEEE 802.11e EDCF simulation model in NS-2.26, technical report TKN-03-019, Telecommunication Networks Group, Technical University Berlin, (2003)
21. IEEE WG 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, *ISO/IEC 8802-11:1999(E),* IEEE Standards 802.11, (1999)
22. IEEE WG 802.11, Part 11: Wireless LAN Medium Access Control (MAC) and Physical layer (PHY) specifications, Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements, IEEE Standards 802.11e, (2005)
23. Executive Briefing: Wireless Network Security, www.docs.hp.com. (2006)

Study on High Performance IPv4/IPv6 Transition and Access Service

Xiaoxiang Leng, Jun Bi, and Miao Zhang

Network Research Center, Tsinghua University
Beijing 100084, China

Abstract. Transition to IPv6 has been recognized as the trend of future Internet. Since a huge amount of resources have been invested on current IPv4-based Internet, how to smoothly transit the Internet from IPv4 to IPv6 is a very important research topic. This paper surveys the state of the art in IPv6 transition, summarizes and compares different IPv6 transition mechanisms and scenarios, discusses the security issues in IPv6 transition, and presents the possible directions for future research.

1 Introduction

The Internet based on IPv4 has made great success in the past 20 years. But mainly due to the scarcity of unallocated IPv4 address, the IPv4 protocol cannot satisfy the requirements of ever expanding Internet. It is reported that the unallocated IPv4 addresses will be used up within 5~6 years [1]. The deployment of NAT can alleviate this problem to some extend, but it breaks the end-to-end characteristic of the Internet, and it can not ultimately resolve such problem as lack of IPv4 addresses. IPv6 protocol suite has been presented in IETF (Internet Engineer Task Force) which uses 128-bit address instead of 32-bit IPv4 address. Transition to IPv6 has been recognized as the most promising direction. This paper presents a comprehensive explanation about the status of current research on IPv6 transition, and indicates the prospect of the future research.

The paper is organized as follows: Section 2 studies on transition mechanisms; Section 3 analyzes typical transition scenario; in Section 4 the security problems in IPv6 transitions are discussed; Section 5 discusses directions of future research; and Section 6 summarizes this paper.

2 Transition Mechanisms

IPv6 transition is a process of gradually replacing IPv4 with IPv6 in the Internet. During the IPv6 transition, network infrastructures and hosts should be upgraded to support IPv6, and network applications should also be migrated to be running in IPv6.

The process of transition to IPv6 will last for a long period. On one hand, the IPv4-based Internet is so diffused that it's impossible to change the whole Internet over one night; On the other hand, the deployment of NAT technology mitigates the urgent need of global IPv4 addresses, and thus delays the deployment of IPv6.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 183 – 194, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The focus in the study of IPv6 transition is changing over the time, from providing network connectivity in which many basic transition mechanisms (NAT-PT, 6to4 etc.) to providing transition schemes for different scenarios during the long transition period. At the same time, the security issues during IPv6 transition also become a hot topic for research.

The research on IPv6 transition can be classified as follows:

(1) Research on basic IPv6 transition mechanisms. A number of different transition mechanisms (e.g., NAT-PT, 6to4, Tunnel Broker, etc.) have been proposed for varied transition requirements. These mechanisms provide tools for the whole transition process. The *ngtrans* WG in IETF has made great efforts on this topic.

(2) Research on analyzing the typical transition scenarios and how to provide relevant transition schemes. As there are a variety of different scenarios during IPv6 transition, the typical scenarios need to be emphasized about IPv6 deployment and applying suitable transition mechanisms. IETF *v6ops* WG and *softwire* WG is now working on this topic.

(3) Research on security issues during IPv6 transition. Security issues during IPv6 transition are always drawing attention. Some security problems are mechanism specific, and some are coming from the coexistence of IPv4/IPv6 [2].

An IPv6 transition mechanism is a method to connect the hosts/networks using the same or different IP protocols under some specific IPv6 transition environment. It's the basis of IPv6 transition. The commonly used transition mechanisms can be divided into three categories: Dual stack, Translation and Tunneling.

With Dual stack method, both IPv4 and IPv6 protocol stacks are deployed on the same node to support both IPv4 and IPv6 protocol.

With Translation method, information and message format are translated between different IP protocols. The pro is that two applications using different IP protocols can communicate with each other. The con is that it breaks the end-to-end characteristic of the Internet, furthermore, it is not scalable in supporting various network applications. There also raises the security problem with Translation method: end-to-end IPSEC encryption can not be exploited with most of Translation methods, consequently exposing to DoS attack on translation gateways. In this way it is not recommended to use translation method in IPv6 transition. NAT-PT, a very popular translation-based technology, has been asked for reconsideration and put into the experimental standard [3].

In IPv6 transition, Tunneling is commonly used for IPv6 hosts/networks to communicate with each other over IPv4 network (i.e., IPv6 over IPv4), and for IPv4 hosts/networks to communicate over IPv6 network (i.e. IPv4 over IPv6). With tunneling methods, the tunnels provide virtual links over the physical network, thus positively having no impact to the upper layer, while leaving the question of dealing with the case that two nodes are with different IP protocols unsolved.

When an IPv6 hosts wants to communicate with countless IPv4 hosts, translation mechanisms should be used.

The dominant translation mechanisms include NAT-PT, BIS, TRT, Socks64 and BIA, which included in three categories: Network Layer Translation, Transport Layer Translation, and Application Layer Translation.

The idea of BIA (Bump-In-the-API) is similar to BIS. In BIA, the difference is that the translation is made between IPv4 APIs and IPv6 APIs. An API translator is inserted between Socket API and TCP/IP modules on the dual stack hosts. The API translator includes three parts: Name Resolver, Address Mapper and Function Mapper. The first two parts are similar to those in BIS. The Function Mapper is in charge of the translation on Socket APIs between IPv4 and IPv6. Unlike other translation mechanisms, BIA achieves the translation without IP header translation. Thus, it will not break the end-to-end security schemes like IPSec.

The commonly used tunneling mechanisms include IPv4/IPv6 configured tunnel, 6to4, ISATAP, Silkroad/Teredo, Tunnel Broker/TSP, DSTM, etc. They can be divided into four categories: IPv6 over IPv4 Tunnel, IPv4 over IPv6 Tunnel, Tunnel traversing NATs and Other Tunnels.

There are still some other scenarios in which the above tunnel mechanisms can not support. Some special tunnel mechanisms are utilized to encapsulate IPv6 packets in some other low layer protocols. These mechanisms include [4]: L2TP, PPTP and PPPoE tunnels at layer 2; PPP-IPv4, IPSec, IPv4-IPv4 tunnels at layer3; TLS/SSL, HTTP and SSH at layer 4; and IPv4 MPLS tunnels, such as 6PE [5].

3 Transition Scenarios

3.1 Typical Transition Scenarios

As there are various scenarios during IPv6 transition, the typical scenarios should be analyzed for more consideration of IPv6 deployment. Four scenarios have been analyzed within IETF *v6ops* WG: ISP Networks, Enterprise Networks, Unmanaged Networks and 3GPP Networks.

ISP Network is a network controlled by some Internet Service Provider. It's composed by two parts: Backbone and Customer Connections. The ISPs need to support IPv4 until the end of IPv6 transition, so they generally use dual stack technology to support IPv6. The troubles in IPv6 transition for ISP Networks mainly come from the fact that Backbone and Customer Connections may be not updated at the same time. One possible requirement is how to connect isolated IPv6 Customer Connections while the Backbone can still only support IPv4. Another possible requirement is how to connect the isolated IPv6 parts of Backbone during the upgrade process of Backbone itself.

Enterprise Network is a network that has multiple internal links, and has one or more router connections to one or more Providers. Normally an enterprise network is managed by a network operations entity. How an enterprise network make IP transition absolutely depends on the requirement of enterprise itself. Possible strategies are: keep existing IPv4 infrastructures unchanged, or widely deploy dual stack equipments, or replace all IPv4 infrastructures with IPv6. One possible problem is how to connect isolated IPv6 hosts and networks while the infrastructures keep using IPv4. Another possible problem is how to provide IPv4 connectivity between internal dual stack hosts while the infrastructure is updated to IPv6.

Unmanaged Network is composed of a single subnet, connected to the Internet through a single Internet Service Provider (ISP) connection via a gateway, which may or may not perform NAT and firewall functions. A characteristic of Unmanaged

Networks is that the gateway is typically not "managed", like the simple Home/Office Networks. The internal hosts generally need not only upgrade to support IPv6 but also keep using IPv4, so Unmanaged Networks would not directly transit to pure IPv6. The transition scenarios are closely related to the IPv6 support of the gateway or the ISP. One major requirement is how to get an IPv6 external connectivity for the gateway through the IPv4 ISP while the local network is upgraded to support IPv6. Another requirement is how to support a dual stack host to communicate with the external IPv6 network while the local gateway is still IPv4 only.

The IPv6 transition in 3GPP (the 3rd Generation Partnership Project) packet data networks can be divided into two parts: transition for GPRS (General Packet Radio Service) network and transition for IMS (IP Multimedia Subsystem). The major requirement is how to connect node/UE (User Equipment) with the same or different IP protocols. As it is decided that IPv6 is the protocol used in IMS, the requirement of IMS transition is how to connect isolated IPv6 networks over IPv4 network.

More efforts should be put on the analysis of possible scenarios. Besides, the current research on transition scenarios is mostly focused on the network connectivity. There should be more attention paid to the support of multicast, anycast, multihoming and mobility in the IPv6 transition.

3.2 Transition Guidelines

The purpose of discussion about different transition scenarios is to design or choose relevant transition schemes [6]. The feasible transition mechanisms should meet the following guidelines:

(1) Scalability. Perhaps the most important consideration is how a given mechanism will scale.

(2) Security. The mechanism should not introduce new security issues, and should not impact the adoption and deployment of IPv6.

(3) Performance. The mechanism should not greatly decrease the performance of existing equipments.

(4) Functionality. In certain transition mechanisms, some of IPv6's "new features" cannot be exploited, and whether to use them need to be decided by the specific of the scenarios

(5) Requirement. The worked mechanisms should be chosen by the requirements of configure method, IP addresses, applications and etc.

(6) Ease of Use. Transition tool configuration should be hidden from the application's end user; if IPv6 is successfully deployed, end users are unlikely to notice the change.

(7) Ease of Management. To introduce a transition mechanism should not bring too much burden of management, and the network during IPv6 transition should be manageable.

3.3 Comparison

Since translation-based methods are commonly not recommended to be used during IPv6 transition, the mostly used transition mechanisms are tunneling-based. The comparison between different tunneling methods is shown in Table 1.

Table 1. Comparison between tunneling methods

	Name	Applicability	Drawbacks
IPv6 over IPv4	IPv6 configured tunnel	IPv6 hosts/islands to communicate with each other or with the native IPv6 network through IPv4 networks.	1.Manual configure
	Tunnel Broker	IPv6 hosts/islands to communicate with each other or with the native IPv6 network through IPv4 networks.	1.Single point of failure 2.Communication bottleneck
	6to4	Isolated IPv6 sites (domains/hosts) attached to an IPv4 network to communicate with each other or with the native IPv6 network.	1.Special 6to4 prefix 2.Difficult to control and management 3.Security threats
	ISATAP	IPv6 hosts inside the IPv4 site to communicate with each other or with the native IPv6 network.	1.Difficult to control and management 2.Security threats
IPv4 over IPv6	IPv4 configured tunnel	IPv4 hosts/networks to connect with each other through IPv6 networks	1.Manual configure
	DSTM	Hosts in native IPv6 network which need to maintain connectivity with hosts/ applications that can only be reached through IPv4	1.Single point of failure 2.Communication bottle- neck
IPv4 over UDP over IPv6	Teredo	Hosts located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over UDP	1.No support for Symmetric NAT
	Silkroad	Hosts located behind one or more IPv4 NATs to obtain IPv6 connectivity by tunneling packets over UDP	1.Single point of failure 2.Communication bottleneck
	TSP	Establish tunnels of various inner protocols (e.g., IPv6, IPv4), inside various outer protocols packets (e.g., IPv4, IPv6, UDP)	1.Single point of failure 2.Communication bottle- neck

From the discussion above, the feasible mechanisms of different transition scenarios can be classified as follows:

(1) Connect isolated large IPv6 networks over IPv4 network, such as the isolated Customer Connections upgraded before Backbone in ISP Networks. The commonly used mechanism is IPv6 configured tunnel.

(2) Connect IPv6/dual stack Islands in IPv4 network to native IPv6 network, such as the Enterprise and Unmanaged Networks with an IPv4-only ISP. In this case IPv6 configured tunnel, Tunnel Broker and 6to4 can be used.

(3) Connect dual stack hosts in IPv4 network to the IPv6 native network, such as dual stack hosts in the Unmanaged Networks with an IPv4-only gateway. The suitable mechanisms are IPv6 configured tunnel, Tunnel Broker and ISATAP. Teredo, Silkroad and TSP can be used if the host is behind one or more IPv4 NATs.

(4) Connect dual stack hosts in pure IPv6 network to IPv4 native network, such as dual stack hosts in Enterprise Networks with pure IPv6 infrastructures. The feasible mechanisms are IPv4 configured tunnel and DSTM.

According to the discussion above, there is no single transition mechanism feasible for all kinds of scenarios. Also there may be several feasible mechanisms for the same scenario. The topic to find available methods providing IPv6 Access Service and ascertain the place of tunnel end point should be paid more attention to. There already has some research on finding the tunnel end point with the help of anycast, DNS, DHCP, PPP and SLP, such as TEP technology.

On another hand, the suitable mechanism for the specific scenario should be decided according to its security and performance characteristics and policies. It is proposed in auto-transition [4] to provide a method to automatically choose the suitable transition mechanism according to the access performance.

Furthermore, the different initialization protocols of different transition mechanisms make the chosen and setup of suitable mechanisms difficult and complex. The IETF *softwire* WG is just set up to define a standard way to discover and setup the soft-wires for connecting the IPv6 networks across IPv4-only network and vice versa. This topic has been divided into two problems: Hub & Spoke and Mesh.

(1) Hub & Spoke. In the situation of Hub & Spoke, the only requirement is to get the external IPv4/IPv6 connection across the IPv6/IPv4-only networks. It's suggested to use L2TPv2 to propagate the softwire information and setup the soft-wires.

(2) Mesh. In the situation of Mesh, not only the connection requirement, but also the routing problem should be considered. It's suggested to use MP-BGP to resolve these problems. However, research on this topic is not enough, more efforts should be made on the interoperation of IPv4 and IPv6.

4 Security Issues in IPv6 Transition

4.1 Translation Based Methods

In translation based methods, the security issues are described as follows:

(1) Impact on the security schemes in IPv6.

Firstly, the translation based methods generally could not support the end-to-end security schemes that depend on the source and destination addresses (e.g., IPSec), because most of them must modify the IP addresses of the packet in the translation except some upper layer mechanisms (e.g., BIA).

Secondly, the encryption schemes of DNS SEC are easily broken by DNS-ALG in the translation of DNS request and reply messages. Therefore the deployment of DNS SEC is also interrupted by the translation based methods.

(2) Potential DoS attacks.

Firstly, since a lot of state information is required to be maintained, it is possible to launch DoS attacks on the translation gateway by sending plenty of small data fragments without any end signal.

Secondly, the attacker can send the translation gateway some packets spoofed the source address as a multicast address to form a reflect-DoS attack.

Thirdly, the characteristic of dynamic binding of translator can also be used to cause the DoS attacks. The buffer and CPU resources of translator may be used out by great deal of messages spoofed as different source addresses in a flash [3].

Generally, the security issues of translation can be mitigated by checking the validness of the addresses, adding authentication schemes and binding statically, but these schemes will greatly consume the system resources, and increase the complexity of these mechanisms. Moreover, the impact on security schemes can not be well settled nowadays.

4.2 Tunneling Based Methods

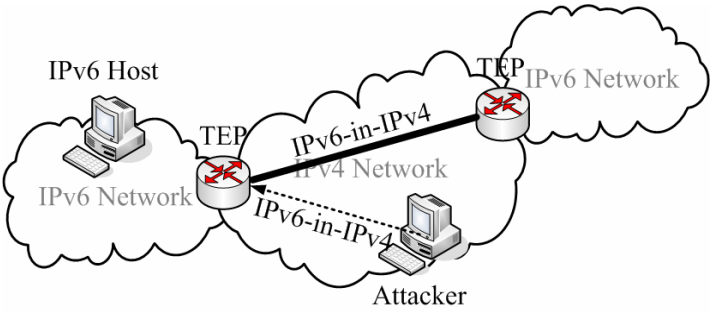


Fig. 1. Security issues of tunneling mechanisms

In tunneling based methods, when a tunnel end point receives an encapsulated data packet, it decapsulates the packet and sends it to the other local forwarding scheme. The security threats in tunneling mechanisms, take IPv6 over IPv4 tunnel for example, are mostly caused by the spoofed encapsulated packet sent by the attackers in IPv4 networks. As shown in Figure 1, the target of attacks can be either a normal IPv6 node or the tunnel end point. These security issues include:

(1) Hard to trace back. The hackers in IPv4 networks can make an attack on the IPv6 nodes through the tunnel end point by sending the spoofed encapsulated packets. It's difficult to trace back in this situation.

(2) Potential reflect-DoS attack. The attackers in IPv4 networks can make a reflect-DoS attack to a normal IPv6 node through the tunnel end point by sending the encapsulated packets with the spoofed IPv6 source address as the specific IPv6 node.

(3) Cheat by IPv6 ND message. Since IPv4 network is treated as the link layer in tunneling technology, the attackers in IPv4 networks can cheat and DoS attack the tunnel end point by sending encapsulated IPv6 ND messages with a spoofed IPv6 link local address [2].

Furthermore, the automatic tunneling mechanisms, such as 6to4 and Teredo, get the information of remote tunnel end point from the certain IPv6 packets. Therefore they would meet some additional security issues:

(1) Attack with IPv4 broadcast address. Take 6to4 [7] mechanism for example, some packets, with destination addresses spoofed and mapped to the broadcast addresses of the 6to4 or relay routers, are sent to the target routers by the attackers in the IPv6 network. In this case, 6to4 or relay routers may be attacked by the broadcast DoS.

(2) Theft of Service. The 6to4 relay administrators would often want to use some policy to limit the use of the relay to specific 6to4 sites and/or specific IPv6 sites. However, some users may be able to use the service regardless of these controls, by configuring the address of the relay using its IPv4 address instead of 192.88.99.1, or using the routing header to route IPv6 packets to reach specific 6to4 relays. (Other routing tricks, such as using static routes, may also be used.). In this way, the 6to4 relay services are thieved and the policies are traversed.

The security issues in tunneling mechanisms can generally be limited by checking the validness of the source/destination addresses at each tunnel end point. But it's hard to deal with the attacks with legal IP addresses now. Since the tunnel end points of configured tunnels are fixed, IPSec can be used to avoid the spoofed attacks [8]. However, there is no effective way to prevent the automatic tunneling mechanisms from DoS/reflect-DoS attacks by the attackers in IPv4 networks.

4.3 IPv4/IPv6 Coexistence

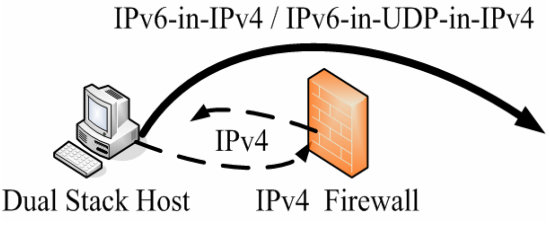


Fig. 2. Traversing IPv4 firewall

The security problems during IPv4/IPv6 coexistence period are mostly related to the break of original IPv4 security schemes [2].

As shown in Figure 2, the attacker can easily traverse the IPv4 filter by an IPv6-in-IPv4 tunnel. In current Internet based on IPv4, firewalls are usually used to protect the information of internal network or limit the internal users. However, during the coexistence period of IPv4 and IPv6, the internal users can traverse the IPv4 firewall to access the external network with no limit by an IPv6-in-IPv4 tunnel. Fortunately, this problem can be resolved by filtering out the packet with protocol 41.

Nevertheless, the IPv6-in-UDP-in-IPv4 tunneling mechanisms (like Teredo, Silkroad, etc.) can also be used to traverse the IPv4 filters. Since the UDP packet, especially with some common used ports, generally can't be discarded, there is no effective way to resolve the UDP traversing problem now. The most probably method is to set the tunnel end point inside local network, and add IPv4 and IPv6 filters at each side of this point. Therefore, with the deployment of IPv6, the IPv6 firewalls should be deployed in time. The technology of IPv6 firewall should be considered as a probably direction of future research.

5 Prospect of IPv6 Transition

5.1 Topology Prospect

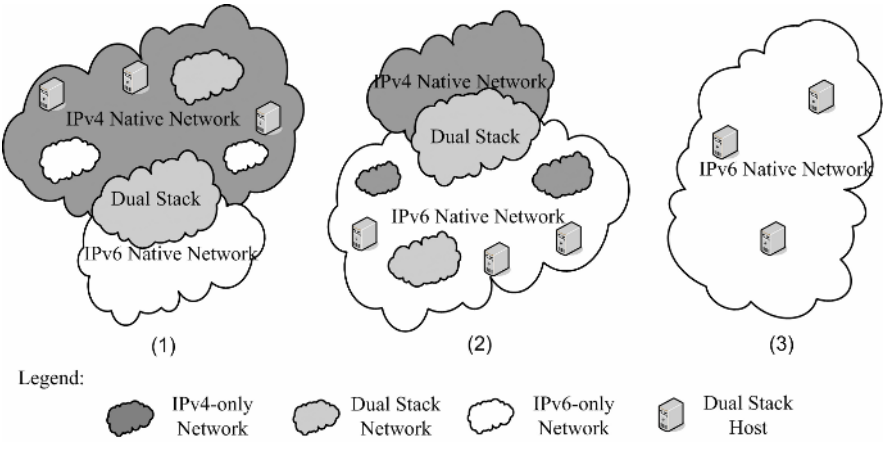


Fig. 3. Three steps of IPv6 transition

The existing transition mechanisms and scenarios are mostly focus on different network topology. The potential generic rules from the research and discussions are:

(1)Make the existing IPv4 network equipments to IPv4/IPv6 dual stack, and keep IPv4 support until the end of IPv6 transition;

(2)Use tunneling technology to connect IPv6 networks isolated by IPv4 network;

(3)Prevent different nodes in the network from talking with each other with different IP protocols, if necessary, upgrade the applications or use the proxy at application layer. It's not recommended to use translation mechanisms.

With the deployment of IPv6, there will be two separated networks: IPv4 Native Network and IPv6 Native Network. The IPv4 Native Network is the legacy of the current Internet where the routers can only forward IPv4 packets, while the hosts may be dual stack by updating the software; The IPv6 Native Network is the combination of new IPv6 networks where the routers can only forward IPv6 packets while the hosts are using dual stack, or only supporting IPv6, and even if the routers are dual stack, there is no global IPv4 address allocated for the network. There are also some dual stack networks. Both routers and hosts have dual stack support, and both global IPv6 and IPv4 address are allocated for the network. Such network can be viewed as the overlapped part of IPv4 Native Network and IPv6 Native Network.

The three-step IPv6 transition with the topology above is analyzed as follows:

(1) Step 1: IPv4 domination, as shown in Figure 3-(1). At the beginning of IPv6 transition, most of existing networks are based on IPv4, and the most important research topic is how to provide IPv6 access service for isolated IPv6 islands. The commonly used mechanism is IPv6 over IPv4 tunnel (like Tunnel Broker, 6to4, ISATAP, etc.). But the current research on basic transition mechanisms mostly

focuses on the connectivity in topology. How to provide scalable, secure and high performance IPv6 access service will be the direction of future research.

(2) Step 2: IPv6 domination, as shown in Figure 3-(2). Along with the deployment of IPv6, IPv6 becomes the domination of Internet. At this time, the dual stack hosts in IPv6 Native Network may need to communicate with the hosts in IPv4 Native Network with the IPv4 applications. The IPv4 over IPv6 transition will be used and this area is still an open research topic.

(3) Step 3: pure IPv6, as shown in Figure 3-(3). Finally, the ISPs gradually stop the support of IPv4, and the network infrastructures have been transited to IPv6 already. There are no global IPv4 networks. However, some legacy IPv4 applications could not be upgraded to support IPv6 because of the lack of source code, or some other reasons. How to make IPv4 legacy applications supported on pure IPv6 infrastructures should also be considered in future.

5.2 Protocol Stack Prospect

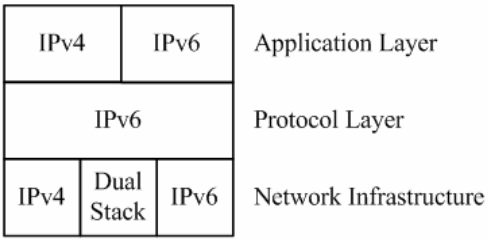


Fig. 4. Univer6 Architecture

In the coexistence of both IPv4 and IPv6 Native Networks, to promote the deployment of IPv6, some important requirements should be addressed:

(1) Protect the legacy investment on IPv4 Native Network.

The routers and switches that can only support IPv4 will not be taken place by IPv6-enabled network devices, due to high cost of new devices and the estimation of no extra income from IPv6 in the near future. End users should have the ability to access IPv6 even with no changes to the IPv4 routers and switches.

(2) Provide a way for universal access.

IPv4 and IPv6 are two different "language" that can not directly talk to each other. There has been a large amount of users in the IPv4 Native Network. With the using up of IPv4 address, it is expected that there will also be a large amount of users in the IPv6 Native Network. The users in IPv4 Native Network and in IPv6 Native Network should have the ability to access each other in an end-to-end way.

(3) Provide support for legacy IPv4 applications.

Even some IPv4 applications can be modified to support IPv6, some will never be modified to have IPv6 support. There should have support for such IPv4 applications to run over IPv6-only networks.

Here we describe architecture Univer6 to meet the requirements analyzed above. The Univer6 architecture is composed of three-layers. In the "Network Infrastructure" layer, there may be IPv4, IPv6 or dual stack. Over the Infrastructure Layer, a "Protocol Layer" takes a place as an overlay network. Because of the lack of IPv4 addresses, IPv4 cannot provide global access ability. The Protocol Layer should use IPv6 to provide universal access for all the end users either in the IPv4 Native Network or in the IPv6 Native Network. Furthermore, the Protocol Layer should support both IPv4 application and IPv6 application in the Application Layer over the IPv6 protocol. There are two key topics on IPv6 transition under the Univer6 Architecture:

(1) How to support both IPv4 and IPv6 applications by IPv6 protocol.

The most different problem with this topic is how to support the legacy IPv4 applications by IPv6 protocol. This has already discussed in Section 6.1. Moreover, some legacy IPv4 applications may need to communicate with the node with IPv6-only applications (i.e., an IPv4-only web browser wants to access the IPv6-only http server). In this situation, it's recommended not to use a translation-based mechanism but a proxy at application layer.

During the IPv6 transition period, dual applications working with both IPv4 and IPv6 are recommended. However, if IP dependencies are required, one of the better solutions would be to build a communication library that provides an IP version - independent API to applications and that hides all dependencies. It could be a possible direction for future research.

(2) How to build overlay IPv6 network on top of different network infrastructures.

This case is similar to how to provide IPv6 access service in different environment. The mechanisms and future research directions are discussed in Section 6.1.

With the use of Univer6 Architecture, the end users can communicate with the people in the IPv6 Native Network and use the service in the IPv6 Native Network no matter which kind of network infrastructure is. The ISPs of IPv4 Native Network are not necessary to replace the IPv4 switches and routers in the near future. Their investments on the IPv4 devices are protected, while their customers can still access IPv6. The ISPs of IPv6 Native Network will increase as more and more people to access the IPv6 Native Network. Therefore, the Univer6 Architecture can satisfy the requirements during IPv6 transition, protect the legacy IPv4 equipments and investments, and accelerate the deployment of IPv6.

6 Conclusion

Due to the prevalence of current Internet, the transition from IPv4 to IPv6 couldn't be accomplished in a short time. How to preserve heavy investments already made, smooth the transition process, and reduce the negative influence to the users and ISPs are the most important tasks of current research on Internet:

(1) Scenario analysis. Typical scenarios analysis is still in progress. Some of them are now in draft mode, such as the Enterprise Network analysis. Other possible scenarios should also be analyzed. Some wireless consideration should also be introduced into the discussion.

(2) Support of multicast, anycast, multihoming and mobility. Both the research on basic transition mechanisms and analysis of typical transition scenarios normally focus on the network connectivity. For the long process of IPv6 transition, the transition mechanisms may not be used only for a while. More efforts should be made on the extension of methods to support multicast, anycast, multihoming and mobility.

(3) Softwire discovery and setup. The different initialization protocols of different transition mechanisms make the chosen and setup of suitable mechanisms difficult and complex. A standard way to discover and setup the soft-wires for connecting the IPv6 networks across IPv4-only network and vice versa is needed for the interoperation of IPv4 and IPv6.

(4) Security consideration. Both the transition mechanisms and the coexistence of IPv4 and IPv6 will introduce more security issues. These problems can not be settled well nowadays. Besides, the IPv6 firewall technology is also a good topic for future research.

(5) Application aspects. Both the network infrastructures and the applications need to support IPv6. The IP version-dependent applications (IPv4-only, IPv4/IPv6 and IPv6-only) make the transition to IPv6 more complex. The IP version-independent application API would be a future research topic. However, how to support the IPv4 legacy applications in IPv6-only networks is still a problem.

References

1. IPv4 Address Report, http://bgp.potaroo.net/ipv4/.
2. Davies, E., Krishnan, S. and Savola, P.: IPv6 Transition/Co-existence Security Considerations, draft-ietf-v6ops-security-overview-03, October 6, 2005.
3. Aoun, C. and Davies, E.: Reasons to Move NAT-PT to Experimental, draft-ietf-v6ops-natpt -to-exprmntl-03, October, 2005.
4. Palet, J., Diaz, M., and Mackay, M.: Evaluation of IPv6 Auto-Transition Algorithm, draft-palet-v6ops-auto-trans-02, October, 2004.
5. Clercq, J., Ooms, D., and Prevost, S.: Connecting IPv6 Islands over IPv4 MPLS using IPv6 Provider Edge Routers (6PE), draft-ooms-v6ops-bgp-tunnel-05, May, 2005.
6. Mackay, M., Edwards, C., Dunmore, M., Chown, T., and Carvalho, G.: A Scenario-Based Review of IPv6 Transition Tools, IEEE Internet Computing, May 2003.
7. Savola P., and Patel, C.: Security Considerations for 6to4, RFC 3964, December 2004.
8. Graveman, R., Parthasarathy, M., Savola, P., and Tschofenig, H.: Using IPsec to Secure IPv6-in-IPv4 Tunnels, draft-ietf-v6ops-ipsec-tunnels-01, August, 2005.

Limiting the Effects of Deafness and Hidden Terminal Problems in Directional Communications

J.L. Bordim[1], T. Hunziker[2], and K. Nakano[3]

[1] University of Brasilia,Dept. of Computer Science,
Campus Universitario, Asa Norte, 70910-900, Brasilia - DF, Brazil
bordim@unb.br
[2] University of Kassel, Dept. of Elect. Engineering., Comm. Lab,
Wilhelmshöher Allee 73, D-34121, Kassel, Germany
hunziker@uni-kassel.de
[3] Department of Information Engineering, School of Engineering,
Hiroshima University, 1-4-1 Kagamiyama, Higashi-Hirhoshima, 739-8527, Japan
nakano@cs.hirhoshima-u.ac.jp

Abstract. It is known that wireless ad hoc networks employing omnidirectional communications suffer from poor network throughput due to inefficient spatial reuse. Although the use of directional communications is expected to provide significant improvements, the lack of efficient mechanisms to deal with deafness and hidden terminal problems makes it difficult to fully explore its benefits. The main contribution of this work is to propose a Medium Access Control (MAC) scheme which aims to lessen the effects of deafness and hidden terminal problems in directional communications without precluding spatial reuse. Unlike other proposals that focus on exploring the characteristics of the physical layer, the proposed MAC protocol relies on simple mechanisms that can be easily coupled with a directional antenna without requiring major modifications to the current MAC standard.

1 Introduction

The nodes in an ad hoc network are usually assumed to share a common channel and to operate with omnidirectional antennas. Since nodes sufficiently apart from each other can communicate simultaneously, one could expect the throughput to improve with the area they cover. However, the relay load imposed by distant nodes and the inefficient spatial reuse provided by omnidirectional antennas results in poor network throughput [2]. Aiming to provide better spatial reuse to increase network capacity, the research community has begun considering ad hoc networks where the nodes are empowered with directional antennas [4,5,6,7,9,10,11,8]. The key benefits provided by directional antennas include reduced co-channel interference, transmission range extension, better spatial reuse and signal quality as compared to their omnidirectional counterparts [3]. Despite of its advantages, developing efficient directional Medium Access Control (MAC)

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 195–206, 2006.
© Springer-Verlag Berlin Heidelberg 2006

protocols in the context of ad hoc networks is a challenging task, not only due to the difficulty in coping with mobility and the absence of a centralized control, but also due to the lack of efficient means to deal with deafness and hidden terminal problems.

The major contribution of this work is to propose a MAC scheme that attempts to minimize deafness and hidden terminal problems in the context of ad hoc networks. Unlike other proposals that focus in exploring the characteristics of the physical layer, the proposed MAC protocol relies on simple mechanisms that can be easily coupled with a directional antenna without requiring major modifications to the current MAC standard.

2 Background and Motivation

Despite the efforts to leverage the capacity of ad hoc networks trough the use of directional communications, a number of problems still remain or need optimized solutions. Indeed, hidden terminals and the wastage of transmission to nodes which are locked away from the transmitter – because the destination is busy sending/receiving to/from other sectors – can have a major impact in the network performance [7]. In what follows, we briefly discuss each of these problems. For this purpose, let us consider the topology shown in Figure 1. To simplify the discussion, we assume that all the packets are sent and received directionally. Suppose that node A is currently communicating directionally with node B. As the directional beam of nodes A and B does not capture nodes C and D, these nodes are free to communicate. Note that, nodes C and D are not aware of the communication between A and B. When using directional communications, the following problems may arise:

1. Suppose that node C wishes to communicate with node A. As node C is unaware of A's activity, it might attempt to reach node A through a directional RTS. Since node A is currently busy, no reply will be received causing node C to increase its backoff window. Eventually, when node A becomes free, node C might have increased its contention window multiple times and, therefore, will have to wait until it expires. This problem is termed as *deafness*, as node A was unable to hear node C's transmission.
2. Let us consider the case in which node C starts talking to node D, while the communication between nodes A and B proceeds. When nodes A and B finish their business, node A might attempt to communicate with node C. In such case, the directional transmission of node A might disrupt the communication between nodes C and D. This problem can be viewed as a special kind of the *hidden terminal problem*.

Although efforts to minimize the above problems have been proposed in the past, the proposed solutions resulted in schemes that are not suitable for ad hoc networks. For example, the protocol proposed in [9] assumes that the mobile devices are able to send and receive messages concurrently. As simultaneous transmission and reception require multiple transceivers – even when

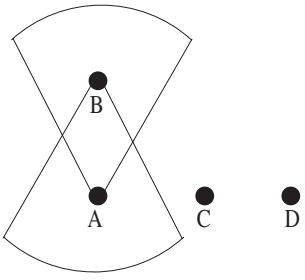


Fig. 1. Example scenario where deafness and hidden terminals problems might occur

sending/receiving through different channels – such approach increases system complexity, cost, antenna size, and power consumption. In the protocol proposed in [10], whenever source and destination nodes complete a communication dialog, they utilize multiple out-of-band tones to inform this event to their neighboring nodes. As pointed out by the authors, there are a number cases in which deafness might still occur. To the best of our knowledge, no efficient MAC protocol capable of coping with both deafness and hidden terminals problems has been proposed in the literature. Obviously, it would be desirable to avoid, or at least minimize, the effects of the aforementioned problems with little overhead and without precluding spatial reuse. Furthermore, it would be interesting to design a MAC protocol with the following properties: (*a*) single transceiver, i.e., no simultaneous transmission and reception; (*b*) simple scheme to associate neighboring nodes to each sector; (*c*) no cross-layer interfaces; and (*d*) litte modifications to the current standard.

In this work we propose a MAC scheme that attempts to minimize deafness and hidden terminal problems in the context of ad hoc networks with the properties mentioned above. In what follows, unless otherwise stated, assume that each mobile terminal is equipped with a switched beam antenna, like the ESPAR (Electronically Steerable Passive Array Radiator) antenna developed at ATR(ACR) [12], and to have similar characteristics as those assumed in [5,6,7,8].

3 Proposed MAC Scheme

In what follows, we begin discussing methods to associate neighboring nodes to a particular sector. Following that, we present a simple MAC scheme which is latter enhanced to limit the effects of deafness and hidden terminal problems.

3.1 Detecting the Direction of Incoming Signals

A number of protocols assume a directional antenna capable of detecting the direction of the incoming signals using *Angle of Arrival* (AoA) techniques [5,9,14]. Other researchers, however, have taken a more conservative approach and designed mechanisms to associate each neighboring node to a particular beam [6,8].

In the former case, the direction of the incoming signals are assumed to be detected during the RTS/CTS dialog. It is worth mentioning that not all directional antennas have such capabilities. In the latter case, nodes are requested to send a special packet while neighboring nodes will *rotate* its directional beam in order to identify the best sector to collect that signal. This special packet could be a tone which would trigger the rotational-sensing and would be long enough to allow neighboring nodes to check all sectors. Once neighboring nodes have identified the best sector for that signal, the source node then sends another packet identifying itself. Neighboring nodes will cache this information and use it whenever needed. In this work we assume that the neighboring-sector association is obtained using either of the above schemes. For further details, we refer the reader to the pertinent references shown above.

3.2 Dual-Channel MAC

Our MAC protocol uses two separated channels namely: a *Control channel* and a *Data channel*. The Control channel is used for exchanging RTS and CTS packets while the Data Channel is reserved for transferring Data and Ack packets. The use of control and Data channels is similar to the channel scheme proposed in [9]. Our protocol, however, does not rely on concurrent transmission and reception. Channel reservation is performed through omnidirectional RTS/CTS exchange between the sender and the receiver. Whenever idle, nodes will be sensing the medium in omnidirectional sensing mode. Suppose that a node S wishes to communicate with node D. Initially, node S senses the Control channel, if sensed busy, then S postpones its RTS transmission. Otherwise, S waits for DIFS period and then enters the backoff phase as in the IEEE802.11 standard [1]. Upon receiving the RTS, node D precedes by carrier sensing and waits for SIFS time before sending the CTS back to S. If node S does not receive a CTS within a timeout period, a new RTS is issued. Again, node S has to go through all the steps of carrier sensing and backoff time before re-transmission.

By employing the mechanisms discussed in Section 3.1, source and destination nodes are able to identify the beam that maximizes the signal strength towards each other during the RTS/CTS exchange. After a successful RTS/CTS exchange, source and destination nodes will leave the Control channel and switch to the Data channel. At this point, sender and receiver will beamform towards each other in order to send/receive Data and Ack packets directionally.

Neighboring nodes, on overhearing the RTS and/or the CTS packets, will set their Directional Network Allocation Vector (DNAV) – which is a directional version of the NAV – towards the direction of the detected signals. Such nodes will defer their own transmissions towards sectors that have set DNAV for the proposed duration of the transfer. However, nodes that have set DNAV are allowed to engage in communication through other sectors, as long as the desired direction of communication does not interfere with any ongoing transfer. In what follows, we refer to the above protocol as a Basic Dual-Channel MAC protocol (or BDC-MAC, for short).

3.3 Limiting the Effects of Deafness and Hidden Terminals

As the above MAC scheme relies on omnidirectional RTS and CTS exchanges, neighboring nodes will be able to learn about the proposed duration of communication thus reducing the effects of deafness. Also, whenever a node sends an RTS to a destination node which is currently receiving a Data packet no collision will occur as two separated channels are used. Furthermore, the implementation of the above protocol is straightforward. However, BDC-MAC cannot prevent collisions from happening at the Data channel as hidden terminal problems are likely to arise. We begin by depicting the occurrence of collision at the Data channel and latter we show how to limit its occurrence.

Figure 2 exhibits a scenario in which collisions at the Data channel are likely to occur. The figure shows an ad hoc network consisting of five nodes where the thin lines represent a link between any two nodes and the numbered arrows represent the communication sequence. The left side of the figure shows a scenario in which two communications have been initiated: the first communication is between nodes C and D and the latter is between nodes A and B. While nodes C and D exchange RTS/CTS packets, neighboring nodes set their DNAV (shown in dark gray) to the direction from which the control packets have been received. As the DNAV of node B does not capture node A, these nodes are allowed to communicate. Note that the RTS/CTS exchange between A and B is performed through the Control channel and will not interfere with the communication of nodes C and D.

While the Data transfer is being carried out between nodes A and B, nodes C and D start a new communication (shown in the right side of the figure). Suppose that node B misses the RTS/CTS exchange between nodes C and D. When the communication of nodes A and B finishes, the previously blocked sectors of node B, which was set towards C and D would have expired. Now, suppose that node B attempts to communicate with node E. As the sector that captures B is not blocked, node E is able to accept the request. When node B starts sending Data over the Data Channel, the directional transfer of node B might interfere with the directional transfer of node C, causing node D to drop the packet.

As described in the example above, even though node E was idle for a long period and its DNAV was accurate, collisions can still occur. Clearly, the traditional DNAV scheme is not enough to prevent collisions at the Data channel. A possible way to limit the occurrence of collisions at the Data channel is to block additional sectors. However, such mechanisms will reduce spatial division which, in turn, might have a negative impact in throughput. The challenge is to devise a mechanism that limits the occurrence of collisions at the Data channel while still being able to provide spatial division. In order to prevent collisions at the Control channel, here we impose some constraints to nodes located within the *lens area* (i.e., the area in which the omnidirectional communications intersect). We assume that each node has two timers: an *idle timer* T_i and a *lens timer* T_l. The T_i timer indicates the amount of time slots a node has been listening to the Control channel, starting from zero up to T_{max}, where T_{max} is maximum duration of a communication. The T_i timer is set to zero whenever a node returns to the Control

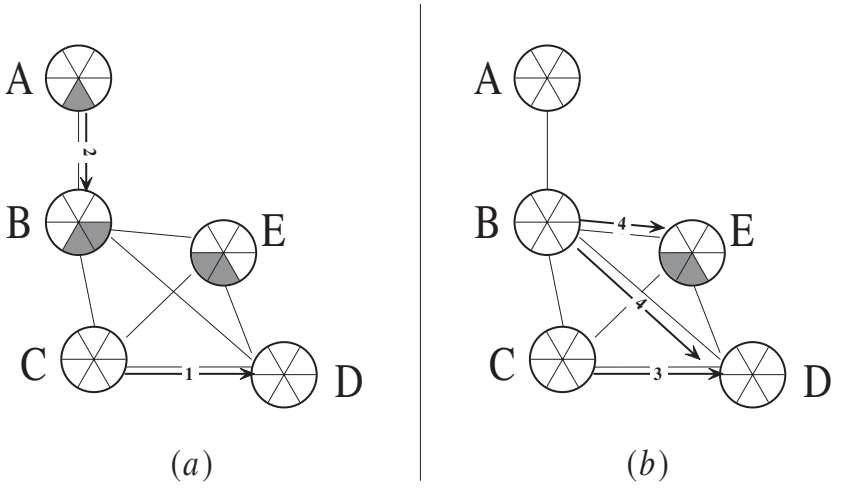


Fig. 2. An example scenario

channel (that is, after a Data transfer). A node will start the T_l timer whenever it finds itself within the lens area, and will reset it at the end of the communication. Let T_l^S, T_i^S, T_l^R and T_i^R denote the sender (S) and receiver (R) timers, respectively. The RTS packet is modified to accommodate the T_i^S and T_l^S timers. Upon receiving an RTS, the destination node performs the following verification when the source node is not in a blocked (DNAV) sector:

Rule 1**.** The destination is within a lens area. The destination node will check whether the source is aware of the communication that created the lens. If $T_i^S \geq T_l^R$ holds, then a CTS can be granted. Clearly, no collision with the communication that created the lens will occur.

Rule 2**.** The destination is outside lens area of the source. If $T_l^S \neq 0$ then check if $T_i^R \geq T_l^S$. If true, then the destination is aware of the communication that created the lens around the sender. The destination can safely reply with a CTS.

Rule 3**.** The lens timer of both source and destination is non-zero. The case where $T_l^S = T_l^R$ is already covered above since $T_i^S \geq T_l^R$ should hold as source and destination must be within the same lens area. If $T_l^S \neq T_l^R$, then the destination has to make sure that the above two conditions are satisfied.

In the previous example, whenever node E finds itself within the lens (created by nodes C and D), it will check whether $T_i^B \geq T_l^E$ holds, before replying to an RTS issued by node B, as required by Rule 1. That is, node E will check whether the node B is aware of the communication that created the lens, so as to prevent collisions at node D. By applying the above rules, whenever a node is within the lens (source, destination, or both), the destination node will be able to decide whether it is safe to accept the request or not. It should be noted

that we allow nodes to communicate when the lens timer of both source and destination is zero. As a result, collisions at the Data channel might eventually occur. To further limit the occurrence of collisions at the Data channel, before transmitting a control packet (RTS and/or CTS), the transmitting node could carrier sense the Data channel towards the direction of the intended receiver. Should a Data packet disruption occur, the node at which the collision occurred will not accept to resume the previous communication until the Data channel has become free once again. This scheme aims to give enough time for those nodes that caused the collision to finish their communication and return to the Control channel. At this point, those nodes which had their communication disrupted, as well as those which caused the disruption, will have to compete for the channel once again, thus limiting further collisions at the Data channel.

It is easy to show that the expected area in which nodes will be setting the lens is $\approx 58\%$ of the area enclosed by the omnidirectional RTS/CTS. Note that these nodes may still be able to communicate when the above rules are satisfied. Therefore, BDC-MAC protocol should be able to provide significant improvements in terms of spatial reuse as compared to a traditional omnidirectional protocol. As we will show in the next section, the above enhancements makes the BDC-MAC resilient to the effects of hidden terminal problems on the Data channel. In addition, our protocol employs simple schemes which can be easily coupled with a directional antenna without the need of major modifications on the current IEEE802.11 standards. Hereafter, the Basic Dual-Channel MAC, including the optimizations proposed above, is denoted simply as Dual-Channel MAC (DC-MAC, for short).

4 Performance Evaluation

The simulations are conduced in QualNet Version 3.6 [13]. QualNet supports the IEEE802.11 DCF MAC as well as directional communications (hereafter, denoted as *Omni* and *Directional* MAC protocols, respectively). These protocols will be used as benchmark in comparing the results. To avoid any interferences that might distort the outcome (e.g., with the choice of a particular routing strategy), in this work we use *static routes*. The directional beam patterns used in the simulations have been obtained from the hardware measurements of the ESPAR antenna [12].

The transmission power used in the simulations is 10dBm and a 2Mbps communication channel is assumed. For DC-MAC we assume that both control and Data channels have similar characteristics. Node mobility is not consider in this work and a *training period* is included in the simulations with enough time for nodes running the Directional protocol to cache the direction of the incoming signals. The averaged results are drawn from twenty runs using different seeds with each run lasting for five minutes. In this work we do not focus on range extension capabilities of the directional antennas. Like the works proposed in [4,5,6], the transmitting node is requested to reduce its transmission power so as to have the same transmission range of an omnidirectional communication.

4.1 Reducing Collisions at the Data Channel

We begin by showing that the mechanisms proposed in Section 3.3 can effectively reduce collisions at the Data Channel. For this purpose, we define the *Data Packet Disruption Rate* (DPDR) as the ratio of Ack packets received over the number of Data packets sent by the source node. Additionally, we define the *Control Packet Loss Ratio* (CPLR) as the ratio of Data packets sent over the number of RTS packets issued for a particular source node. In other words, the CPLR accounts for the unsuccessful RTS/CTS exchange while the DPDR shows the percentage of Data packet disruption after a successful RTS/CTS exchange.

In what follows, we utilize the example scenario shown in Figure 2, where nodes A and C are selected as the source nodes for nodes E and D, respectively, with node B serving as relay for node A. The application layer at nodes A and C are set to generate CBR traffic at the rate of 340Kbps and 680Kbps, respectively. To simplify the analysis, we utilize an *ideal* antenna in this set of simulations (i.e., a pattern without side and back lobes). Figure 3 shows the results in terms of DPDR and CPLR, for nodes C and D with DC-MAC and BDC-MAC. Recall that BDC-MAC does not include the optimizations discussed in Section 3.3, which have been incorporated to reduce the occurrence of collisions at the Data channel. For comparison purposes, the results for Omni and Directional protocols are also shown. The figure shows that nearly 14% of the Data packets sent by node C are dropped with BDC-MAC. This is consistent with the DPDR of the Directional protocol in which more than 12% of the Data packets issued by node C are lost. That is, even after a successful RTS/CTS exchange between nodes C and D, the Directional and BDC-MAC protocols cannot guarantee that the Data packets will be safely delivered. By implementing the mechanisms proposed in Section 3.3, DC-MAC can significantly reduce collisions at the Data channel. Although collisions still arise, the DPDR for DC-MAC is comparable to that of the Omni protocol.

As the lens and idle timers are managed at the MAC layer, it is possible that the physical layer hands an RTS packet to the MAC layer just after the lens timer has expired. To see this, suppose that node D is sending an Ack packet back to node C. At the same time, node B, unaware of the communication between nodes C and D, might start sending an RTS to node E. In case the physical layer at node E hands the RTS to the MAC layer when the lens timer has expired, node E would be in a position to reply. Note that if nodes C and D miss to receive the RTS packet, their lens timers would not be set. In such case, collisions at the Data channel might arise at a later stage with DC-MAC. A simple solution to this problem would be to extend the lens timer by a certain threshold. Another possibility is to include the Source and Destination MAC addresses in the CTS packet such that nodes would be able to set the lens timers even if only the CTS packet is received. These solutions, however, increase the amount of time spend on the Control channel. We note that collisions at the Data channel due to the above occurrences are infrequent, less than 0.2% of the Data packets in this scenario. Thus, in this work we have chosen not to extend the lens timers or to include additional information to the CTS.

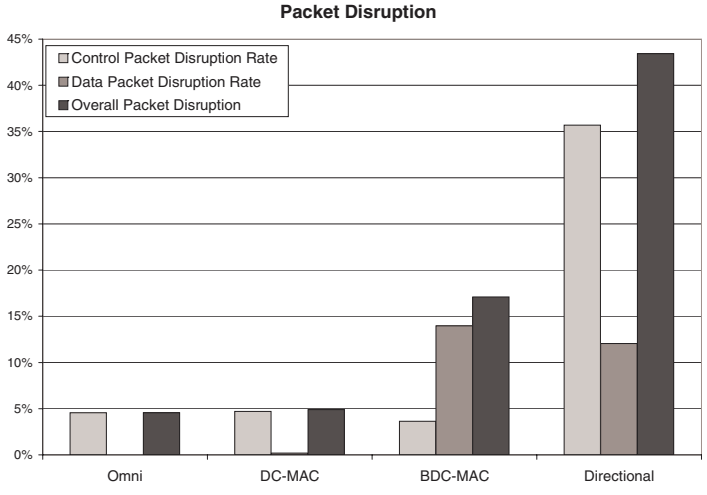


Fig. 3. Average packet disruption for DC-MAC and BDC-MAC

In view of the fact that both DC-MAC and BDC-MAC rely on omnidirectional RTS/CTS exchange, their CPRL values are comparable with the Omni protocol ($\approx 5\%$). Due to hidden terminal problems, the CPDR for the Directional protocol surpasses 35%, which is more than 7 times higher than the Omni protocol. The overall packet disruption, that is, the ratio of the Data packets received at node D over the number of RTS packets issued by node C is also shown. The figure shows that more than 43% of the packets issued by node C are lost when using the Directional protocol and $\approx 17\%$ for BDC-MAC. For Omni and DC-MAC, this value is less than 5% with the majority ($\approx 99.8\%$) of the packet loss occurring during the RTS/CTS exchange.

4.2 Deafness and Hidden Terminal Problems in Grid Network Topology

It is known that directional communications do not perform well in string topologies due to deafness and hidden terminal problems [7]. As the proposed protocol aims to reduce their effects, our goal in this subsection is to verify the performance, in terms of throughput and end-to-end delay, of the proposed MAC protocol in such topologies.

Here we consider a grid of size 3×9. Node separation is fixed at 260 meters. Nodes $n_{1,1}$, $n_{2,1}$, and $n_{3,1}$ are selected as the source nodes and the average throughput and delay performance is verified at every two hops. The Data packet sizes used are: 512, 1024 and 1536 Bytes of CBR traffic, generated at a rate of 2, 4, and $6ms$, respectively.

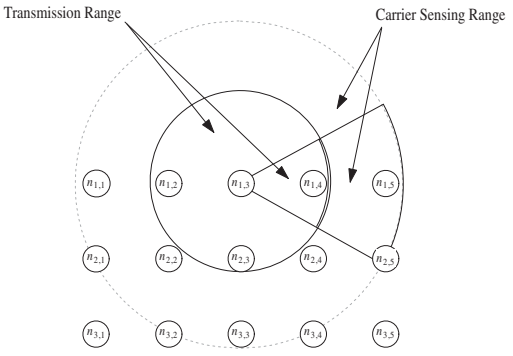
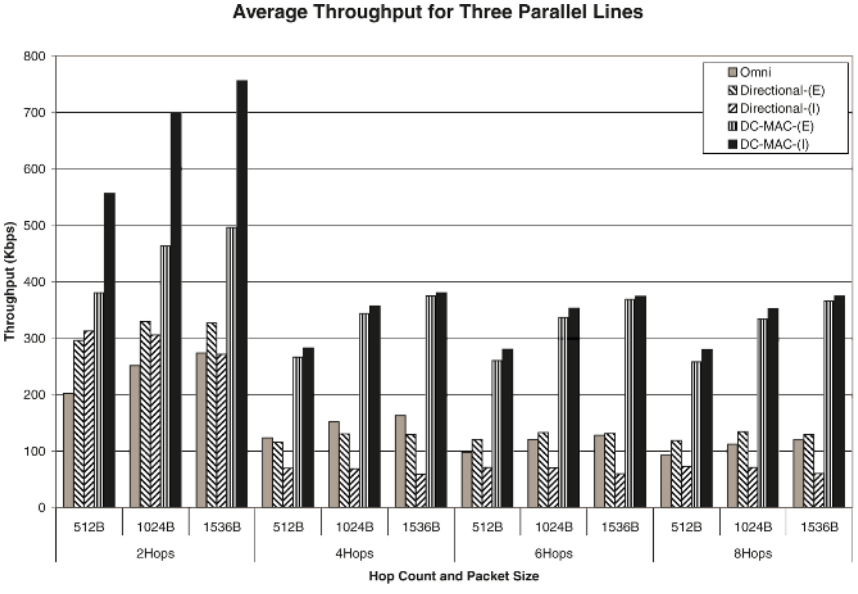


Fig. 4. Transmission and carrier sense range for omnidirectional and directional communication with an ideal antenna

Figure 5(a) shows the average throughput results for three parallel lines with Omni, Directional, and DC-MAC protocols (in the figure, DC-MAC(E) and DC-MAC(I) stands for DC-MAC with ESPAR and Ideal antenna, respectively). The reduced carrier sensing area provided by directional communications facilitates the Directional protocol to deliver a better performance than the Omni protocol. As the number of hops increases, the effects of neighboring activity make the performance of the Directional protocol to deteriorate. As shown in Figure 4, when nodes $n_{1,3}$ and $n_{1,4}$ are talking with the Directional protocol, node $n_{2,5}$ would not be able to communicate with node $n_{2,4}$. That is, the directional beam of node $n_{1,3}$ captures the directional beam that node $n_{2,5}$ would use to reach node $n_{2,4}$. When using the Directional protocol with an ideal beam, node $n_{2,4}$ would find the channel towards $n_{2,5}$ idle. Should node $n_{2,4}$ try to reach node $n_{2,5}$, the directional packet issued by $n_{2,4}$ would probably be dropped. With a broader beam, such as that of the ESPAR antenna, node $n_{2,4}$ might be able to carrier sense neighboring activity and hence reduce the effects of deafness. For this reason, the Directional protocol utilizing the ESPAR antenna performs better than with the ideal antenna.

Although carrier sensing also has a negative impact on the performance of DC-MAC, this impact is not as severe as in the Omni and Directional protocols. We also note that DC-MAC is able to take advantage of the sharper beams of the ideal antenna to deliver a better throughput. This is clear in the case of 2 hops. With an increase in the number of hops traversed, the throughput difference reduces significantly between DC-MAC(E) and DC-MAC(I) due to the reasons explained above. Nevertheless, the throughput performance of DC-MAC is ≈ 3 times the throughput of the Omni protocol and Directional protocols.

The average end-to-end delay is shown in Figure 5(b). As can be observed, DC-MAC has a much lower average end-to-end delay as compared to the Omni and Directional protocols – less than 50% at $6-8$ hops with ESPAR and less than 30% with an ideal antenna.



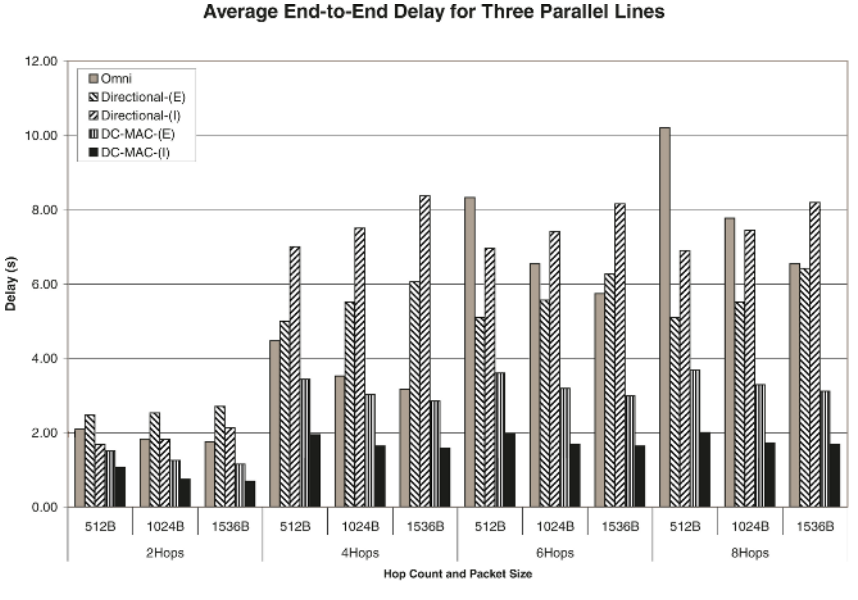


Fig. 5. (*a*) Average throughput for three parallel lines at different hop count and Data packet size. (*b*) Average end-to-end delay for three parallel lines at different hop count and Data packet size.

References

1. Institute of Electrical and Electronics Engineers (IEEE), *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Standard 802.11, June 1999, http://standards.ieee.org.
2. P. Gupta and P. R. Kumar, *The capacity of wireless networks*, IEEE Transactions on Information Theory, Mar 2000, Volume: 46, Issue: 2, pp. 388-404.
3. T. Rappaport, *Wireless Communications: Principles and Practice*, Prentice Hall PTR, Upper Saddle River, NJ, 2001.
4. Y.-B. Ko, V. Shankarkumar and N. H. Vaidya, *Medium Access Control Protocols Using Directional Antennas in Ad Hoc Networks*, IEEE INFOCOM'2000, March 2000.
5. A. Nasipuri, S. Ye, and R. E. Hiromoto, *A MAC Protocol for Mobile Ad Hoc Networks Using Directional Antennas*, Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC 2000), Chicago, September 2000.
6. S. Bandyopadhyay, K. Hasuike, S. Horisawa, S. Tawara, *An Adaptive MAC Protocol for Wireless Ad Hoc Community Network (WACNet) Using Electronically Steerable Passive Array Radiator Antenna*, Proc of the GLOBECOM 2001, November 25-29, 2001, San Antonio, Texas, USA.
7. R. R. Choudhury, X. Yang, R. Ramanathan, and N. Vaidya, *Using directional antennas for Medium Access Control in Ad hoc Networks*, ACM International Conference on Mobile Computing and Networking MobiCom, September 2002.
8. J. L. Bordim, T. Ueda, and S. Tanaka, *Delivering the Benefits of Directional Communications for Ad Hoc Networks Through an Efficient Directional MAC Protocol*, in Proc. of the 11th International Conference on Telecommunications (ICT-2004), pp. 461-470, Aug. 1-6, 2004, Fortaleza, Brazil.
9. Zhuochuan Huang, Chien-Chung Shen, C. Srisathapornphat, and C. Jaikaeo, *A Busy-Tone Based Directional MAC Protocol for Ad Hoc Networks*, in IEEE MILCOM, Anaheim, CA, 2002.
10. R. R. Choudhury and N. Vaidya, *Deafness: A MAC Problem in Ad Hoc Networks when using Directional Antennas* in Proce. of the 10th IEEE International Conference on Network Protocols, Berlin, October, 2004.
11. H. Gossain, C. M. Cordeiro, D. Cavalcanti, and D. P. Agrawal, *The Deafness Problems and Solutions in Wireless Ad Hoc Networks using Directional Antennas*, in IEEE Workshop on Wireless Ad Hoc and Sensor Networks, November 2004.
12. T. Ohira and K. Gyoda, *Electronically steerable passive array radiator antennas for low-cost analog adaptive beamforming*, in Proceedings of the IEEE International Conference on Phased Array Systems and Technology, 2000. pp: 101-104.
13. Scalable Networks Technologies, *QualNet - Network Simulator*, http://www.scalable-networks.com/. 2
14. Y. Wang and J. J. Garcia-Luna-Aceves, *Collision Avoidance in Single-Channel Ad Hoc Networks Using Directional Antennas*, in Proc. of IEEE Intl. Conf. on Distributed Computing Systems, May 2003.2

Enforcing Dimension-Order Routing in On-Chip Torus Networks Without Virtual Channels

Hiroki Matsutani[1], Michihiro Koibuchi[2], and Hideharu Amano[1]

[1] Keio University, 3-14-1, Hiyoshi, Kohoku-ku, Yokohama, JAPAN 223-8522
{matutani, hunga}@am.ics.keio.ac.jp
[2] National Institute of Informatics (NII)
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, JAPAN 101-8430
koibuchi@nii.ac.jp

Abstract. In the case of simple tile-based architecture, such as small reconfigurable processor arrays, a virtual-channel mechanism, which requires additional logic and pipeline stages, will be one of the crucial factors for a low cost implementation of their on-chip routers. To guarantee deadlock-free packet transfer with no virtual channels on tori, we propose a non-minimal strategy consistent with the rule of dimension-order routing (DOR) algorithm. Since embedded streaming applications usually generate predictable data traffic, the path set can be customized to the traffic from alternative DOR paths. Although the proposed strategy does not use any virtual channels, it achieves almost the same performance as virtual-channel routers on tori in eleven of 18 application traces.

1 Introduction

Networks-on-Chips (NoCs) have been studied to connect a number of modules in a chip by introducing a network structure similar to that in parallel computers[1,2]. They are utilized not only for high-performance computing but also for small embedded streaming applications, such as JPEG or Viterbi coders, mostly used in consumer equipments.

Two-dimensional mesh[3,4,5] and torus[2] have been employed as a typical on-chip interconnect, because their regular arrangement is intuitively matched to the two-dimensional VLSI layout. A simple and popular deterministic routing on such networks is dimension-order routing (DOR), which simply routes packets in y-dimension after completing in x-dimension. Although a torus network has twice bisection bandwidth of a same-sized mesh, DOR originally requires two virtual channels to avoid deadlocks in the case of 2-D tori. Whereas in meshes, DOR is deadlock-free without virtual channels. A virtual-channel mechanism strongly affects the router architecture, and as reported in [6], it is possible to drastically reduce the hardware amount of on-chip routers by removing a virtual-channel mechanism from them.

In recent design methodologies, embedded systems and their applications are designed with system level description languages like System-C, and simulated in the early stage of design. The task distribution is statically decided in this stage, and the amount of traffic between nodes can be analyzed. Routing techniques according to the analysis of traffic patterns have been researched. In this paper, we propose a non-minimal strategy for DOR to guarantee deadlock-freedom without virtual channels in

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 207–218, 2006.
© Springer-Verlag Berlin Heidelberg 2006

tori, according to records of traffic analysis in the design stage of SoCs. Additionally, when application traffic patterns are partially unknown, or incompletely analyzed, which hardly occur, we use a simple mechanism to eject and reinject some packets at a few intermediate nodes, so as to break their cyclic dependencies[7].

Notice that virtual channels play a key role to avoid not only deadlocks, but also head-of-line blocking. However, average hop count is small in the cases of stream processing, so the advantage of throughput is not fully extracted in our target NoCs.

This paper is organized as follows. Existing deadlock-free solutions without virtual-channels to route packets are surveyed in Section 2. The virtual-channel free routing strategy is proposed in Section 3 and 4. In Section 5, for the evaluations, our routing strategy was applied to 18 real application traces and the results are compared with a virtual-channel router on tori. Finally, We conclude in Section 6.

2 Related Work

To remove virtual channels from DOR routers on tori, the bubble flow control[8] has been developed mainly for parallel computers. This is an injection limitation mechanism that guarantees continuous message movement in the network by preserving at least one empty packet in routers' channel buffer. Thus, the bubble flow control is used with virtual-cut through (VCT) switching that requires enough buffers to always store the largest packet. However, since the buffer size is a crucial factor for implementing lightweight routers, NoCs usually employ wormhole switching, not VCT switching.

Another technique to remove cyclic dependencies is the packet reinjection at a few intermediate nodes[7]. That is, packets that could introduce cyclic dependencies are once ejected from network and reinjected so as to cut the cyclic dependencies. This technique enables minimal routing algorithms to be implemented on system area networks (SANs) with no virtual channels, such as Myrinet. Although this technique requires infinite buffers for deadlock-freedom in theory, it is not a problem on SANs because of the large main memory available in each host PC. However, the size of buffer memory is strongly limited in NoCs, so this method would not be directly applied into NoCs.

3 Virtual-Channel Free Routing Strategy

3.1 Dimension-Order Non-minimal Routing

In 1-dimensional tori, DOR provides two alternative paths to reach a destination: a path using a wrap-around channel and the other one not using the wrap-around channel. Thus, there are up to 2^n alternative paths when DOR is used in n-dimensional tori. Figure 1 shows four alternative paths from node $N_{(0,2)}$ to $N_{(2,1)}$ in a 4×4 torus. In Figure 1(a), packets are routed by using $x+$ channels followed by $y-$ channels. In the same way, (b) uses $x-$ followed by $y-$ channels, (c) uses $x+$ followed by $y+$ channels, and (d) uses $x-$ followed by $y+$ channels. In this example, (a) and (b) take minimal paths, but (c) and (d) introduce non-minimal paths. Because non-minimal paths waste the network bandwidth and increase latency, DOR implementation commonly uses only

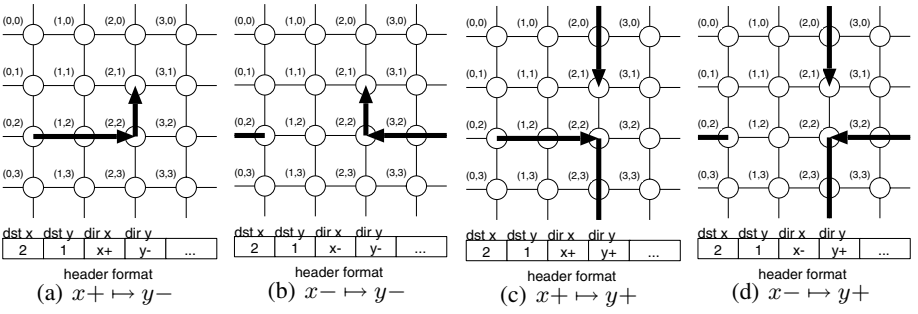


Fig. 1. Routing paths provided by DONR on a 4×4 torus. In each header format, the coordinate of destination and the traveling directions are represented as "dst" and "dir".

minimal paths. When minimal alternative paths are available like (a) and (b), only a single path is randomly selected[9].

In tori, cyclic dependencies can be formed inside a dimension, and a virtual-channel mechanism has been originally used to remove them. In this paper, we remove a virtual-channel mechanism from conventional torus routers in order to reduce the amount of hardware for routers. In order to make deadlock-free path sets for tori without virtual channels, we use a small amount of non-minimal paths with DOR by analyzing the communication pattern of the target application (the path selection strategy is described in Section 3.2). To distinguish the path sets from path hops on DOR strategy, we call such a routing "**dimension-order non-minimal routing (DONR)**".

3.2 Path Selection

A path selection strategy is designed to satisfy the following policies:

1. Deadlock-freedom is satisfied on tori without virtual channels.
2. The number of non-minimal paths is minimized.
3. Computations of the path set are completed within a few seconds.

The proposed strategy searches the best set of routing paths that achieves deadlock-freedom on tori without virtual channels and minimizes the use of inefficient non-minimal paths from all the possible combinations of alternative paths provided by DONR. Additionally, to obtain the solution within a realistic time frame, the path search algorithm employs efficient pruning techniques.

Terminology. Each node in a k-ary n-cube is denoted as N_i, where $i = \{0, ..., k^n - 1\}$. Figure 2 is an example of a 4×4 torus. The set of unidirectional links along a given direction forms a unidirectional cycle. The unidirectional cycle that includes node N_i for $x+$ direction is denoted as ring R_i^{x+}. The other unidirectional cycles for the $x-$, $y+$, and $y-$ directions are denoted as ring R_i^{x-}, R_i^{y+}, and R_i^{y-}, respectively. In Figure 2, ring R_0^{x+} is a unidirectional cycle $N_0 - N_1 - N_2 - N_3 - N_0$.

Path Search Algorithm. For a given communication pattern, this algorithm searches a deadlock-free routing path-set under DONR on tori without virtual channels. When all

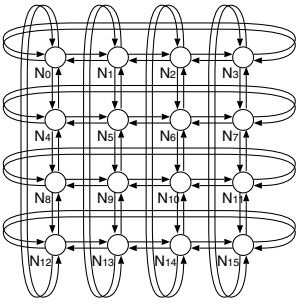


Fig. 2. An example of 2-D torus

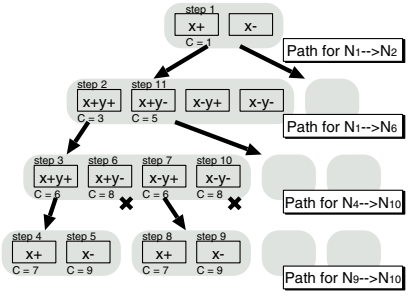


Fig. 3. A search tree for DONR paths

source-destination pairs select the paths that do not use wrap-around channels, the routing path-set becomes the same as that in mesh, which does not require virtual channels. Thus, at least one virtual-channel free solution always exists. However, since such a mesh-like path set will drastically reduce its performance, the proposed algorithm carefully assigns non-minimal paths to a few source-destination pairs in order to minimize the impact of inefficient non-minimal paths. Actually, assigning a non-minimal path to the source-destination pair that transfers a large amount of data will waste interconnection's bandwidth and cause serious performance degradation. On the other hand, the negative impact of a non-minimal path on a source-destination pair with a small data transfer is considered to be slight. In the proposed algorithm, non-minimal paths are preferentially assigned to the pairs that transfers smaller data amount.

Each set of routing paths is evaluated by the following cost function.

$$Cost = \sum_{s=0}^{n-1} \sum_{d=0}^{n-1} H_{(s,d)} \times D_{(s,d)} \tag{1}$$

where n is the number of nodes, $D_{(s,d)}$ is the total amount of communication data from node N_s to N_d, and $H_{(s,d)}$ is the path hop count from node N_s to N_d on a target torus. Based on this cost function, source-destination pairs that transfer a large amount of total data can take minimal paths in most cases. On the other hand, non-minimal paths are sometimes assigned to pairs with a small amount of data transfer so as to remove cyclic dependencies and mitigate the performance degradation involved by unavoidable inconvenient paths. The set of deadlock-free routing paths whose cost is minimum is the best solution. The algorithm that confirms deadlock-freedom on a given set of routing paths is presented below.

Since a search tree of all the possible routing path-sets is huge, it is difficult to find the best set of routing paths in a realistic time frame (a few seconds). Actually, there are up to $2^{nk^n(k^n-1)}$ sets of routing paths in a k-ary n-cube when all-to-all communications are used[1]. In this paper, we employ the branch-and-bound method as a pruning technique of the search tree.

[1] This is true only when all source-destination pairs take 2^n alternative paths. When both source and destination nodes' coordinates on one dimension are same, the number of alternative paths is decreased to $2^{(n-1)}$. Thus, the actual number of possible path-sets is less than $2^{nk^n(k^n-1)}$.

Now we illustrate the path search algorithm. Figure 3 shows an example of a search tree for a 2-D torus. In this example, a path is assigned for each source-destination pair $N_1 \mapsto N_2$, $N_1 \mapsto N_6$, $N_4 \mapsto N_{10}$, and $N_9 \mapsto N_{10}$. There are up to four alternative paths ($x+y+$, $x+y-$, $x-y+$, and $x-y-$) for each source-destination pair. Notice that there are only two alternative paths in the cases of $N_1 \mapsto N_2$ and $N_9 \mapsto N_{10}$, because packets can be sent by using only x-dimensional channels. The cost of each branch (denoted as C in the figure) is evaluated by Equation 1. For the sake of simplicity, the amount of data transfer D is set to *one* in all source-destination pairs equally. The path search starts from step 1 in this figure. In step 4, path $x+$ is allocated for pair $N_1 \mapsto N_2$, path $x + y+$ is for $N_1 \mapsto N_6$, path $x + y+$ is for $N_4 \mapsto N_{10}$, and path $x+$ is for $N_9 \mapsto N_{10}$. Since the minimum cost of the solutions is updated to *seven*, all the branches whose cost is greater than *seven* are trimmed away after step 4. Thus, the branch in step 6 is pruned. In the same way, the branches that could form deadlocks are pruned away. By cutting branches, computational efforts to find the minimum solution can be drastically reduced.

Cycle Detection Algorithm. This function confirms that a given path-set P is deadlock-free on a torus.

1. (Initialize bitmaps.) A bitmap of all nodes is allocated for each direction ($x+$, $x-$, $y+$, $y-$) as shown in Figure 4.
2. (Mark nodes.) For all pairs of source N_s and destination node N_d in path set P :
   (a) When data amount $D_{(s,d)}$ is equal to $zero$, return to step 2.
   (b) A path from node N_s to N_d on the torus is denoted as $P_{(s,d)}$. For example, $P_{(4,14)} = \{N_4, N_5, N_6, N_{10}, N_{14}\}$.
   (c) From path $P_{(s,d)}$, a source node, a destination node, and turning nodes that change the traveling direction are removed. For path $P_{(4,14)}$, source node N_4, destination node N_{14}, and turning node N_6 are removed. Thus, $P'_{(4,14)} = \{N_5, N_{10}\}$.
   (d) For all nodes in $P'_{(s,d)}$, the corresponding node on the bitmap of its traveling direction is marked. In the case of $P'_{(4,14)}$, node N_5 is marked on the $x+$ bitmap and N_6 is marked on the $y+$ bitmap (Figure 4).
3. (Detect cycles.) When all nodes on a ring are not marked, no cycle is formed on the ring. We confirmed this with the following Theorem and Proof. In Figure 4, cycles are formed at ring R_{15}^{x-} and R_{13}^{y-}.
4. (Judge.) When no cycle is formed on all rings in the torus, deadlock-freedom is guaranteed on the path set P without virtual channels.

Theorem . *When all nodes on a ring are not marked, no cycle is formed on the ring.*

Proof. The Theorem noted above is proved with ring R_i^{x+}. It can be proved with other rings in the same way. A source node, a destination node, and turning nodes are not marked in the bitmaps. Thus, when node N_i is marked on the $x+$ bitmap, there is at least one packet that occupies the $x-$ channel of N_i until the $x+$ channel of N_i is released. When all nodes on the ring are not marked, there is at least one channel in which no packet is waiting for its release. A cycle on the ring is broken by such channels. Thus, when all nodes on a ring are not marked, no cycle is formed on the ring. □

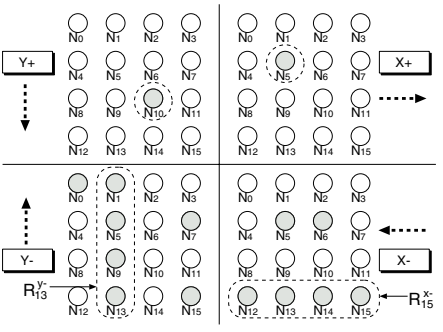


Fig. 4. Bitmap $x+$, $x-$, $y+$, and $y-$ for the cycle detection

Fast Computation Techniques for Path Selection. Following two techniques are applied to achieve the fast path selection strategy.

- In the search tree (e.g., Figure 3), source-destination pairs that transfer a large amount of data are placed just below the root of the tree, in order to prune the fruitless branches in the earlier phases of the search.
- To shrink the search tree, assigning non-minimal paths for source-destination pairs with a large data transfer is gradually limited according to the acceptable time-frame for the path selection. Thus, a part of them can always take minimal when the time limitation is strict.

4 An Extension for Dynamic Traffic

We extend the virtual-channel free routing strategy to avoid deadlocks in the case of including the dynamic traffic as follows: an intermediate node temporarily stores packets to a certain destination node on its local memory, and reinjects them to their destination, as briefly introduced in Section 2[7]. Since the packet is once ejected from the network by being stored in the intermediate node's local memory, cyclic dependencies across the intermediate node can be removed.

Here, we introduce a method for identifying ejection-and-reinjection packets required by deadlock-free operation at each intermediate node from all of dynamic packets. The method takes the following steps:

1. The routing strategy shown in the previous section is applied only for the static traffic, which can be completely pre-analyzed. Cyclic channel dependencies on such static traffic can be cut by making at least one channel used by no packets (*non-static-traffic channel*) on every ring.
2. Step (1) indicates that only dynamic traffic passes through the non-static-traffic channels. In order to break the cyclic channel dependency on a ring caused by dynamic traffic, a node connected to the non-static-traffic channel performs ejection-and-reinjection only for the dynamic traffic that will pass through the neighboring non-static-traffic channel.

As discussed in Section 2, applying this method into NoCs originally may introduce the following problems: 1) a packet reinjection operation newly introduces relatively large delay at an intermediate node, and 2) an infinite buffer is needed for temporarily storing packets at intermediate nodes, because starting a reinjection of a packet before finishing its ejection cannot cut the cyclic dependency around the intermediate node[7].

To resolve the former problem, we use a cut-through based switching, that allows a header flit to be sent if the sender has enough buffer to store its receiving remain flits. Even if the header flit is blocked at the next router, all body flits will be able to be stored at the host. Thus, this cut-through based switching, which drastically improves the latency at intermediate host especially at long packets, does not disturb the ejection-and-reinjection operation. Since a host usually has larger buffers than those of wormhole routers, it is feasible for NoCs.

Regarding the latter problem, all nodes are potentially expected to provide a buffer enough for packetization and de-packetization at their network interface, and this buffer can be used to temporarily store the dynamic traffic. However, an infinite buffer is needed at each intermediate node so that a whole packet is always stored. Since infinite memory cannot be provided in NoCs, some dynamic packets cannot be temporarily stored in an intermediate node's local memory when its buffer is filled with dynamic packets waiting for their reinjection. In this case, such overflowed packet is discarded at the intermediate node, and a NACK packet must be sent to its original source node so as to request a retransmission of the discarded packet.

This method simply extends end-to-end flow control mechanism on the higher-level protocol on NoCs; the NACK packet is possibly to be issued at intermediate nodes. Such higher-level protocols on NoCs have been widely investigated for various purposes[1,2], and enabling "sophisticated network protocol stacks on a simple hardware" is one of key features of NoCs. The proposed method can be fit with such NoCs.

5 Performance Evaluation

5.1 Simulation Environments

A flit-level simulator written in C++ was used for measuring throughput on DOR+v1, DOR+v2, and DONR+v1 routers. A simple model consisting of input buffers, a crossbar, and arbiters of the crossbar is used for the switching fabric in the router. A header flit requires at least three clock cycles to be transferred to the next router or core; one cycle for the routing decision, one cycle for transferring a flit from an input channel to an output channel through a crossbar, and the remaining cycle for transferring the flit to the next router or core. Wormhole switching is used as a switching technique on the router. A node injects a packet independently of each other, and we set packet length for 16 flits including one header flit.

In the proposed routing strategy, since traffic pattern is an important performance factor, we used real application traces. As a typical stream application, we use the communication pattern on a Soft-Input Soft-Output (SISO) Viterbi decoder. The target systems are consisting of 16 tiles, each of which can process a task assigned to the tile. Its task flow graph and mapping result are shown in Figure 5.

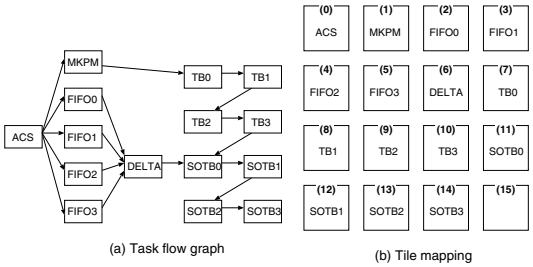


Fig. 5. Mapping result for a Viterbi decoder

In addition, we used application traces captured from NAS Parallel Benchmark (NPB) programs, because they would enable us to evaluate with various sizes / patterns of real application traffic. We selected BT, SP, CG, MG, and IS programs from NPB. The class of problem is set to "W", and the numbers of nodes for solving the problems are 9, 16, 32, 36, and 64.

For DONR+v1, a deadlock-free path set is generated by using the proposed routing strategy. The calculation time for finding a feasible path set on each application trace is shown in Table 1. The required time is less than a few seconds in most cases, except for IS traces including all-to-all communications that significantly widen the search space.

Table 1. Calculation time for DONR's paths. (@ Intel Pentium4 2.6GHz)

Application	Time (sec)	Application	Time (sec)	Application	Time (sec)
Viterbi (16 node)	0.005	BT (9 node)	0.014	BT (16 node)	0.030
BT (36 node)	4.798	BT (64 node)	1.018	SP (9 node)	0.015
SP (16 node)	0.011	SP (36 node)	5.404	SP (64 node)	0.130
CG (16 node)	0.011	CG (32 node)	0.196	CG (64 node)	0.298
MG (16 node)	0.011	MG (32 node)	5.354	MG (64 node)	0.427
IS (16 node)	0.865	IS (32 node)	19.901	IS (64 node)	3600.000

5.2 Simulation Results on Static Traffic

DONR+v1 is compared with DOR+v1 and DOR+v2 with the 18 application traces in terms of throughput and latency. We firstly simulate them assuming that whole traffic patterns can be known. Simulation results on mixing the static and dynamic traffic are shown in Section 5.3.

Figure 6 shows the throughput (accepted traffic) versus the latency with the Viterbi trace on the three router-types. The average hop counts on each router are also shown in the parenthesis. As shown in the graph, the DOR+v2 router, which can exploit wrap-around channels, achieves higher throughput than that of the DOR+v1. Although DONR+v1 does not have virtual channels, it can fully exploit wrap-around channels without introducing non-minimal paths. In this Viterbi trace, each task is manually mapped onto nodes so as to shorten the average hop count. Most communications are limited between neighboring nodes in the stream processing, and such one-hop

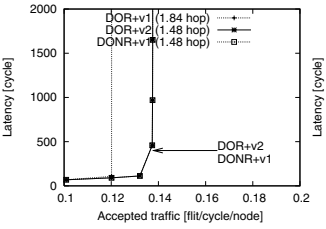


Fig. 6. Viterbi (16n)

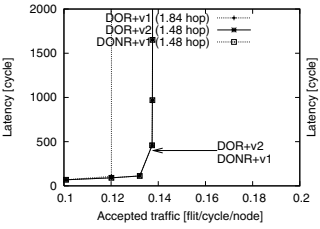


Fig. 7. BT (9n)

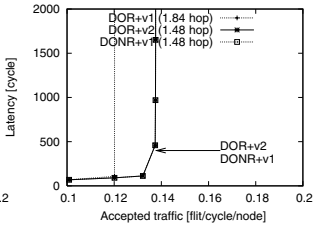


Fig. 8. BT (16n)

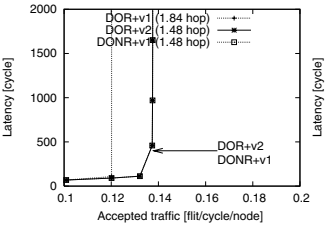


Fig. 9. BT (36n)

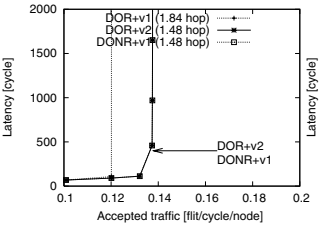


Fig. 10. BT (64n)

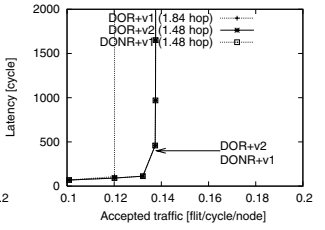


Fig. 11. SP (9n)

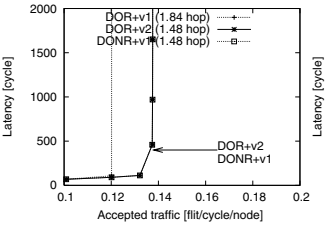


Fig. 12. SP (16n)

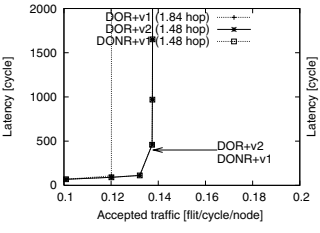


Fig. 13. SP (36n)

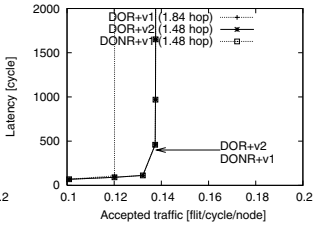


Fig. 14. SP (64n)

communications do not form a cycle. Hence, the simple DONR+v1 router can achieve deadlock-freedom without non-minimal paths, and thus its performance is the same as a virtual-channel router.

We shift to the evaluation results on NPB traces. Figure 7-10 show the results with BT program on 9-, 16-, 36-, and 64-node networks, respectively. In these BT traces, the average hop counts are relatively small, regardless of the network size. Therefore, the DONR+v1 router can fully exploit wrap-around channels and achieve almost the same throughput as the DOR+v2 router.

Figure 21-23 show the results with IS program on 16-, 32-, and 64-node networks, respectively. Since the IS program is dominated by all-to-all communications, it is difficult for the DONR+v1 to remove virtual channels without introducing a number of non-minimal paths. Actually, in the 32- and 64-node traces, the average hop counts of the DONR+v1 are as long as those of DOR+v1, and their performance is not so different from a conventional mesh router. On the other hand, the 16-node program on the DONR+v1 router can be routed without non-minimal paths, though its performance

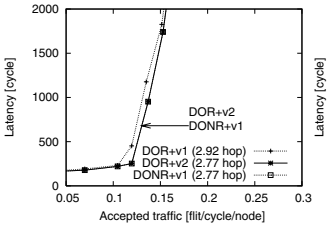


Fig. 15. CG (16n)

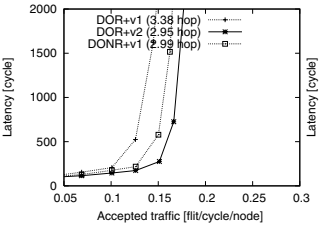


Fig. 16. CG (32n)

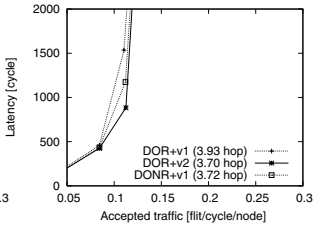


Fig. 17. CG (64n)

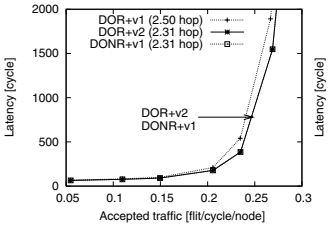


Fig. 18. MG (16n)

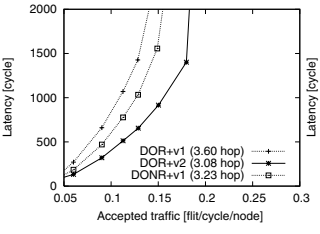


Fig. 19. MG (32n)

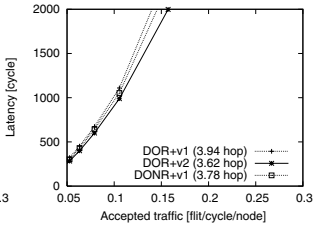


Fig. 20. MG (64n)

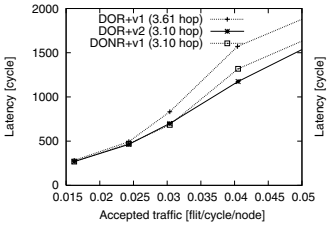


Fig. 21. IS (16n)

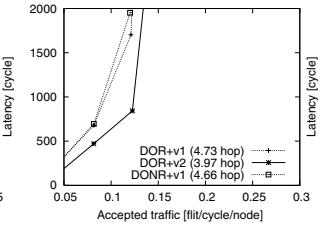


Fig. 22. IS (32n)

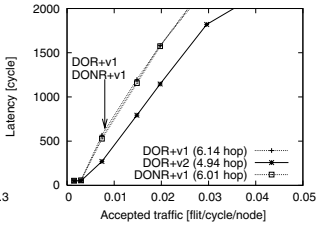


Fig. 23. IS (64n)

is slightly inferior to the DOR+v2. This is because wrap-around channels are used by only one-hop-per-dimension packets, which introduce no deadlocks, in the case of 16-node.

5.3 Simulation Results on Static and Dynamic Traffic

To confirm that the proposed strategy can be used with unpredictable dynamic traffic, the extension for dynamic traffic is evaluated assuming that certain degrees (0%, 25%, 50%, and 100%) of the total traffic cannot be analyzed. Since the size of intermediate nodes' buffer for reinjection affects the performance, we simulate them with different buffer sizes: buffers with capacities for 2 packets (b2) and 16 packets (b16).

Figure 24 shows results of the 16-node BT program in which different degrees (0%, 25%, 50%, and 100%) of the total traffic are not statically analyzed. As shown in the graph, there is no difference between each degree of unpredictable packets. Additionally, the results of Viterbi and SP traces have similar characteristics (not shown in the

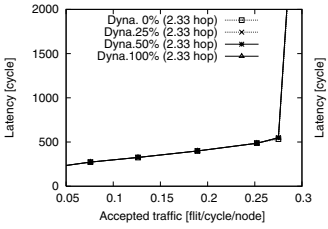


Fig. 24. BT+dyna (16n,b2)

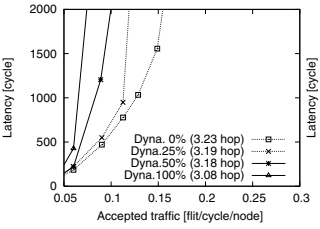


Fig. 25. MG+dyna (32n,b2)

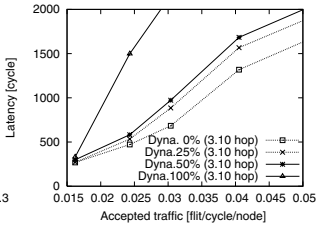


Fig. 26. IS+dyna (16n,b2)

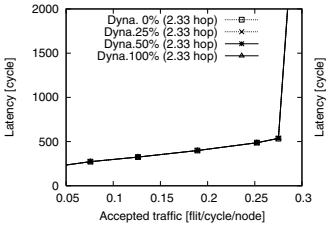


Fig. 27. BT+dyna (16n,b16)

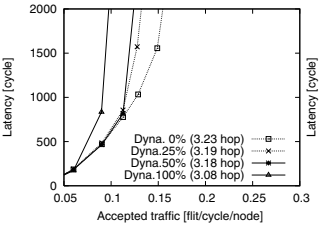


Fig. 28. MG+dyna (32n,b16)

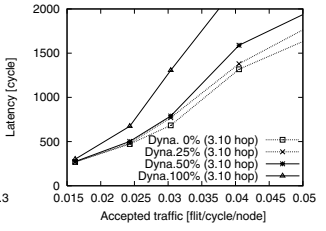


Fig. 29. IS+dyna (16n,b16)

figures). Since these traces contain a lot of neighboring communications that do not cause deadlocks, the packet reinjections are infrequent.

Figure 25 shows the results of the 32-node MG program. We can see that the performance is reduced down corresponding to the number of reinjections, and thus Dyna.0% is the best among them. Figure 28 shows the results with eject-and-reinjection buffers of 16 packets (b16). As shown in the graph, the performances of Dyna.25%, 50%, and 100% are improved compared with those with smaller buffer b2. This relatively large buffer can process more eject-and-reinjecting packets, resulting in smaller number of packet discards and retransmissions.

It is rarely possible that communication patterns cannot be known in stream processing. By using the extension for dynamic traffic, the proposed virtual-channel free strategy can be used for the case of mixing the static and dynamic traffic. However, in such cases (it is rare), the performance is reduced down corresponding to the number of eject-and-reinjections and retransmissions in a part of applications as shown above.

6 Conclusions

A scheme for removing a virtual-channel mechanism from DOR routers on tori is accomplished by the following steps: 1) introducing non-minimal DOR (DONR), and 2) a path selection strategy to analyze a communication pattern and to find a deadlock-free path set without virtual channels.

Since embedded streaming applications usually generate predictable data traffic, the path set is customized to the traffic from alternative paths with the DOR rule. When application traffic patterns are incompletely pre-analyzed, only a part of unpredictable

packets are ejected and reinjected at some intermediate nodes in order to avoid deadlocks. In the case, we extend end-to-end flow control so as to implement this mechanism with a limited size of node buffers. Simulation results show that the generated virtual-channel free path-set can achieve almost the same performance as that on a virtual-channel router in eleven of the 18 application traces. For the other traces, the performance is reduced corresponding to the number of non-minimal paths introduced for virtual-channel freedom. Additionally, we confirmed that the proposed strategy could be used with unpredictable dynamic traffic by applying a simple mechanism to reinject some packets at a few intermediate nodes. The calculation time for generating a virtual-channel free path-set is negligible in most of traces, and thus we can easily apply the proposed strategy for application-specific on-chip torus networks.

Acknowledgments

This work was supported by Joint Research Fund, "Network-on-Chip Architecture," National Institute of Informatics.

References

1. Benini, L., Micheli, G.D.: Networks on Chips: A New SoC Paradigm. IEEE Computer **35**(1) (2002) 70–78
2. Dally, W.J., Towles, B.: Route Packets, Not Wires: On-Chip Interconnection Networks. In: Proceedings of the Design Automation Conference. (2001) 684–689
3. Burger, D., et. al.: Scaling to the End of Silicon with EDGE Architectures. IEEE Computer **37**(7) (2004) 44–55
4. Liang, J., et. al.: An Architecture and Compiler for Scalable On-Chip Communication. IEEE Transactions on Very Large Scale Integration Systems **12**(7) (2004) 711–726
5. Taylor, M.B., et. al.: The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. IEEE Micro **22**(2) (2002) 25–35
6. Matsutani, H., Koibuchi, M., Amano, H.: A Virtual-Channel Free Mapping for Application-Specific On-Chip Torus Networks. In: Proceedings of the International Conference on Parallel and Distributed Computing Systems. (2006) 24–31
7. Flich, J., Lopez, P., Malumbres, M.P., Duato, J.: Boosting the Performance of Myrinet Networks. IEEE Transactions on Parallel and Distributed Systems **13**(7) (2002) 693–709
8. Puente, V., Beivide, R., Gregorio, J.A., Prellezo, J.M., Duato, J., Izu, C.: Adaptive Bubble Router: A Design to Improve Performance in Torus Networks. In: Proceedings of the International Conference on Parallel Processing. (1999) 58–67
9. Dally, W.J., Towles, B.: Principles and Practices of Interconnection Networks. Morgan Kaufmann (2004)

A Distributed Backbone Formation Algorithm for Mobile Ad Hoc Networks

Orhan Dagdeviren and Kayhan Erciyes

Izmir Institute of Technology
Computer Eng. Dept., Urla, Izmir 35340, Turkey
{orhandagdeviren, kayhanerciyes}@iyte.edu.tr

Abstract. Construction of a backbone architecture is an important issue in mobile ad hoc networks(MANET)s to ease routing and resource management. We propose a new fully distributed algorithm for backbone formation in MANETs that constructs a directed ring architecture. We show the operation of the algorithm, analyze its message complexity and provide results in the simulation environment of ns2. Our results conform that the algorithm is scalable in terms of its running time and round-trip delay against mobility, surface area, number of nodes and number of clusterheads.

1 Introduction

MANETs do not have any fixed infrastructure and consist of wireless mobile nodes that perform various data communication tasks. MANETs have potential applications in rescue operations, mobile conferences, battlefield communications etc. Clustering has become an important approach to manage MANETs. In large, dynamic ad hoc networks, it is very hard to construct an efficient network topology. By clustering the entire network, one can decrease the size of the problem into small sized clusters. Clustering schemes can be classified as Dominating Set(DS)-based, low-maintanence, mobility-aware, energy-efficient, load-balancing and combined-metrics-based clustering [1]. DS-based clustering algorithms [2,3,4,5,6] like Wu's CDS(Connected Dominating Set) algorithm [2], Chen's WCDS(Weakly Connected Dominating Set) algorithm [3], Dominating Set Based Clustering Algorithm [4] try to find a DS for a MANET so that the number of mobile nodes that participate in route search can be reduced. Low-maintenance clustering [7,8,9,10] schemes aim at providing stable cluster architectures for upper-layer protocols with little cluster maintenance costs. Mobility-aware clustering [11,12,13] takes the mobility behavior of mobile nodes into consideration. Energy-efficient clustering [14,15,16] manages to use the battery energy of mobile nodes wisely in a MANET. Load-balancing clustering schemes [14,17,18] attempt to limit the number of mobile nodes in each cluster to a specified range so that clusters are of similar size. Combined-metrics-based clustering [19] usually considers multiple metrics, such as node degree, cluster size, mobility speed and battery energy in cluster configuration, especially in clusterhead decision [1].

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 219–230, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Load-balancing clustering schemes like Merging Clustering Algorithm(MCA) [17], Adaptive Multi-hop Clustering [18] (AMC) and Degree-Load-Balancing Clustering (DBLC) [14] distribute the workload of a network more evenly into clusters by limiting the number of mobile nodes in each cluster in a defined range. But the weakness of these algorithms is the lack of virtual backbone formation to serve the lower layer protocols like routing, or the upper layer operating system services like distributed *mutual exclusion* protocol [20]. In this study, we propose a backbone formation algorithm for load-balancing clustering algorithms where backbone is constructed as a ring architecture by directing clusterheads in a minimum spanning tree to each other. Related work in this area is reviewed in Section 2, we define, illustrate and analyze our algorithm in Section 3, provide implementation results in Section 4 and the final section provides the conclusions drawn.

2 Background

MCA finds clusters in a MANET by merging the clusters to form higher level clusters as mentioned in Gallagher, Humblet, Spira's algorithm [21]. The clustering operation we apply, however, operates by discarding the minimum spanning tree. This reduces the message complexity from *O(nlogn)* to *O(n)*. Upper and lower bound heuristics for clustering operation are used which result in a balanced number of nodes in the cluster formed. AMC maintains multihop cluster structure as similar to MCA. For cluster maintenance, each mobile node periodically broadcasts its information, its id, cluster id and status to others within the same cluster. Clusters are obtained by merging, and upper and lower bounds are used for controlling the cluster size. DLBC periodically runs the clustering scheme in order to keep the number of nodes in each cluster approximately equal to a system parameter, *ED*, which indicates the optimum number of mobile nodes that a clusterhead can handle. A clusterhead degrades to an ordinary member node if the difference between *ED* and the number of mobile nodes that if currently serves exceeds some value, *Max_Delta* [1]. As mentioned, load-balancing algorithms partition the network into a balanced number of clusters but a backbone is not constructed.

Wu et al.'s CDS Algorithm is a step wise operational distributed algorithm, in which every node has to wait for others in a lock state. In this algorithm, nodes exchange neighbor list messages to decide marking process. Algorithm has two phases of marking operation to find a connected dominating set. A CDS with small size reduces the number of nodes involved in routing-related tasks. Further heuristics and degree checking functionalities are added in Dominating Set based Clustering Algorithm to find the minimal CDS. The number of clusters produced by the CDS clustering is rather large and the cluster structure is highly overlapping [1]. Chen proposed a WCDS scheme by relaxing the requirement of direct connection between neighboring dominating nodes. Backbone formation is supported by the construction of CDS or WCDS in these algorithms, but adjusting the cluster size is not mentioned.

3 Our Algorithm

3.1 General Idea of the Algorithm

The algorithm we propose constructs a backbone architecture on a clustered MANET. Different than other algorithms, the backbone is constructed as a directed ring architecture to gain the advantage of this topology and to give better services to other middleware protocols such as *distributed mutual exclusion* [20] and *total order multicast*. The second contribution is to connect the clusterheads of a balanced clustering scheme which completes two essential needs of clustering by having balanced clusters and minimized routing delay. Besides these, the backbone formation algorithm is fault tolerant as the third contribution.

3.2 Description of the Algorithm

We assume that the MANET is partitioned by a load-balanced clustering algorithm like MCA, AMC or DLBC. Each node has distinct *node_id*, knows its *clusterhead_id* are the basic assumptions of our algorithm as well as these clustering algorithms.

Our main idea is to maintain a directed ring architecture by constructing a minimum spanning tree between clusterheads and classifying clusterheads into *BACKBONE* or *LEAF* nodes, periodically. To maintain these structures, each clusterhead broadcasts a *Leader_Info* message by flooding. In this phase, clustermember nodes act as routers to transmit *Leader_Info* messages. Algorithm has two modes of operation; hop-based backbone formation scheme and position-based backbone formation scheme. In hop-based backbone formation scheme, minimum number of hops between clusterheads are taken into consideration in a minimum spanning tree construction. Minimum hop counts can be obtained during flooding scheme. For highly mobile scenarios, an agreement between clusterheads must be maintained to guarantee the consistent hop information. In position-based backbone formation scheme, positions of clusterheads are used to construct the minimum spanning tree. If each node knows its velocity and the direction of the velocity, these information can be appended with a timestamp to the *Leader_Info* message to construct a better minimum spanning tree. But in this mode, nodes must be equipped with a position tracker like a GPS receiver.

Every node in the network performs the same local algorithm. The finite state machine of the algorithm is shown in Fig. 1. Each node can be either in *IDLE*, *BACKBONE* or *LEAF* states described below.

- *IDLE*: Initially all clusterheads are in *IDLE* state. If *Period_TOUT* occurs, each clusterhead broadcasts a *Leader_Info* message to the destination node and will make a state transition to *WT_INFO* state. If *Leader_Info* message is received, the clusterhead makes a state transition to *LEAF* state and reconstructs the ring by reorganizing the minimum spanning tree.
- *WT_INFO*: A clusterhead in *WT_INFO* state waits for *Leader_Info* message. If a *Leader_Info* message is received, the clusterhead makes a state

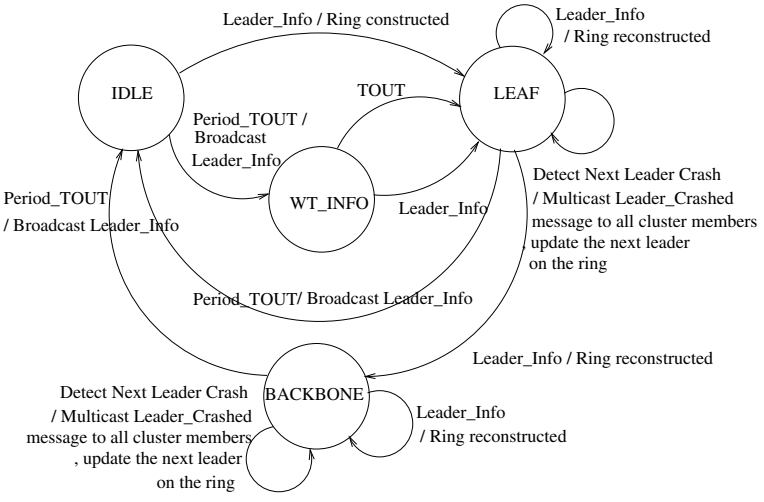


Fig. 1. Finite State Machine

transition to *LEAF* state and reconstructs the ring. If *TOUT* occurs, clusterhead makes a transition to *LEAF* state which indicates that the network has only two active partitions.

- *LEAF*: A clusterhead in *LEAF* state has a degree of 1 in its local minimum spanning tree. If a *Leader_Info* message is received, the clusterhead reconstructs the ring and makes a state transition to *BACKBONE* state if the degree exceeds 1. If *Period_TOUT* occurs, clusterhead makes a transition to *IDLE* state to restart the backbone formation.
- *BACKBONE*: A clusterhead in *BACKBONE* state has a degree greater than 1. For each *Leader_Info* message received, the ring is reconstructed. If *Period_TOUT* occurs, the backbone formation is restarted.

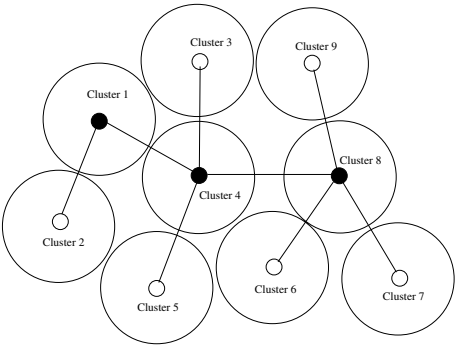


Fig. 2. MANET with its minimum spanning tree

```
1.Procedure ring_construct
2.begin
3.  construct minimum spanning tree by total received leader information
4.  if my degree is equal to 1
       execute ordinary_leaf
9.  else
10.     set my state to BACKBONE
    if I am a BACKBONE leader or a LEAF leader which can't find next leader
        execute backbone_proc
15.end
```

Fig. 3. Procedure executed by all leaders to construct a Ring Architecture

A balanced clustered MANET with its clusterheads and minimum spanning tree is shown in Fig. 2. *BACKBONE* clusterheads are shown as black and *LEAF* clusterheads are shown as white nodes. The main part of the algorithm is the construction of a ring architecture by orienting clusterheads in the minimum spanning tree. General idea is to divide the ring into two parts. A directed path of *BACKBONE* clusterheads and a directed path of *LEAF* nodes. Finally, these two directed paths are connected to each other to maintain the ring architecture. Each clusterhead aims to find the next clusterhead(leader) to construct the ring architecture by the procedure in Fig. 3.

Our first aim is to form the vital part of the backbone. The *BACKBONE* clusterheads are directed to each other from starting *BACKBONE* clusterhead to the end. Starting *BACKBONE* clusterhead is the one with the smallest connectivity to other *BACKBONE* nodes. This selection policy of *BACKBONE* clusterhead results in smaller hops and reduced routing delay. Ending *BACKBONE* clusterhead is directed to its *LEAF* with the smallest *node_id*.

LEAF leaders firstly execute the procedure in Fig. 5 to find the next leader on the ring. The aim of directing *LEAF* leaders with the same *BACKBONE* leaders to each other is to make the routing process over the same *BACKBONE* leader to reduce delay. *LEAF* leaders which can't find the next leader execute the procedure in Fig. 4 and search for a *LEAF* leader from the previous BACKBONE leaders of their parent to find a *LEAF* leader. Our last aim is to connect the *LEAF* leaders of different *BACKBONE* parents to maintain the routing operation by using the *BACKBONE* leaders.

Third contribution of our algorithm is the fault tolerance of clusterheads. Each clusterhead can maintain the list of cluster member nodes in load-balancing algorithms like MCA, AMC or DLBC. In our backbone formation algorithm, this list can be appended to *Leader_Info* message by each clusterhead. After the formation of the ring is completed, if a clusterhead detects the crash of the next clusterhead, it can multicast a *Leader_dead* message to all cluster members which initiates clustering operation. To support this functionality, clustering layer must be updated. If this crash occurs during a real time operation,

```
1.Procedure backbone_proc
2.begin
3.      find the starting BACKBONE leader such that its connectivity to
        other BACKBONE nodes is smallest between all other BACKBONE
        leaders.
4.      find the next leader of starting BACKBONE.
5.      If next leader found
6.        set the temporary BACKBONE leader to next leader of starting
          BACKBONE.
7.      If not found
8.        find LEAF leader with smallest node_id of starting BACKBONE leader.
9.       mark the starting BACKBONE leader.
10.     if I am starting BACKBONE leader set my next leader to found
        value
11.     else
12.         while all BACKBONE nodes are not marked
13.           find the next BACKBONE leader of temporary BACKBONE leader
              with smallest distance which is not marked.
14.           if found
15.              set the next leader of temporary BACKBONE leader to found
                 value
16.              mark the temporary BACKBONE leader
17.              set the temporary BACKBONE leader to next leader
18.            else
19.              set the next leader of temporary BACKBONE leader
                 to LEAF with smallest node_id.
20.              mark this LEAF leader
21.         if I am a LEAF leader which can't find next leader
21.         find a child with smallest node_id from a previous BACKBONE
            leaders of my parent BACKBONE leader.
23.         if found set the next leader
24.         else set the next leader to starting BACKBONE leader
25.end
```

Fig. 4. Procedure executed by BACKBONE leaders and LEAF leaders which can't find next leader

```
1.Procedure ordinary_leaf_proc
2.begin
3.      set my state to LEAF
4.      Find a LEAF leader with same parent and nearest greater node_id.
5.      If found
6.         set my next leader to this LEAF leader's node_id and mark
           this LEAF.
7.end
```

Fig. 5. Procedure executed by LEAF leaders

clusterhead updates its next leader to next-next leader and continues its operation since it knows the global information of all clusterheads.

3.3 An Example Operation

Assume the MANET with clusterheads(leaders) in Fig. 6.a. Clusters are obtained using MCA. Nodes 65, 15, 98, 30, 40, 13, 28, 80, 74, 19, 51 and 99 are the leaders of clusters 1 to 12, respectively. Each clusterhead floods the *Leader_Info* message to the network. After each clusterhead receives the *Leader_Info* message of the others, minimum spaning tree in Fig. 6.a is constructed by all clusterheads. Nodes 65, 98, 40, 13, 80, 19 and 99 identify themselves as *LEAF* leaders since their degrees are all 1. Nodes 15, 30, 28, 74 and 51 identify themselves as *BACKBONE* leaders since their degrees are greater than 1. *BACKBONE* leaders are filled with black and *LEAF* leaders are filled with white as shown in Fig. 6.a.

To connect the *BACKBONE* nodes, a starting *BACKBONE* leader must be chosen. The criteria is to select the *BACKBONE* node which has the smallest connection to other *BACKBONE* leaders. Node 15 is connected to 30, 30 is connected to 15 and 28, 28 is connected to node 30 and node 74, node 74 is connected to node 28 and 51, 51 is connected to 74. Node 15 and 51 can be the choice for starting *BACKBONE* leader. 15 is selected because its *node_id* is smaller than 51. 15 selects the next leader as 30, 30 selects the next leader 28, operation continues in this way. The ending *BACKBONE* leader directs to its *LEAF* with the smallest *node_id*. These directions can be seen in Fig. 6.b with bold directed lines.

LEAF leaders of a *BACKBONE* leader are directed to each other from smallest to greatest. Node 19 is directed to 99, 13 is directed to 80, 65 is directed 98 as seen in Fig. 6.c with dotted directed lines.

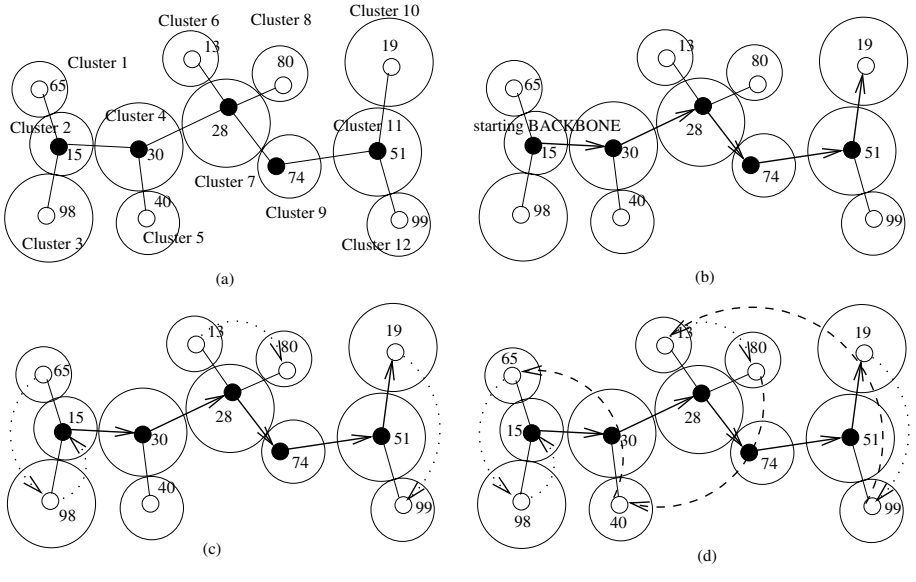


Fig. 6. An Example Operation

Lastly, *LEAF* leaders of different *BACKBONE* leaders are connected as in Fig. 6.d. Each *LEAF* leader which can not find the next leader, searches for a *LEAF* leader from the children of the previous *BACKBONE* leader of its parent *BACKBONE* leader. 99 is connected to 13, 80 is connected to 40, 40 is connected 65, 98 is connected to 15 shown with dashed lines in Fig. 6.d.

3.4 Analysis

Theorem 1. *Message complexity of the backbone formation algorithm is* $O(Kn)$.

Proof. Assume that we have n nodes in our network. K leaders flood the message to the network. Total number of messages in this case is Kn which means that message complexity has an upper bound of $O(n)$.

Theorem 2. *Time complexity of the backbone formation algorithm is* $O(Kn)$.

Proof. Assume that we have n nodes in our network. Flooding of K messages to the network takes Kn time.

4 Results

We implemented the distributed backbone formation algorithm with the *ns*2 simulator. Clustering is obtained using the MCA algorithm. Cluster size can be adjusted by the K heuristic of MCA. Position-based backbone formation algorithm is implemented.

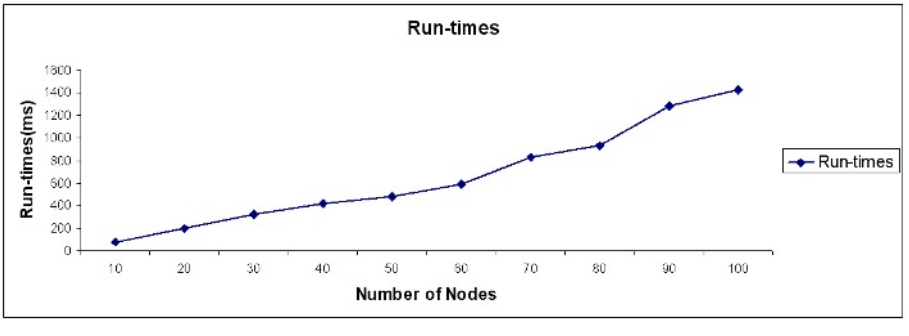


Fig. 7. Runtime Performance

Different size of flat surfaces are chosen for each simulation to create medium, small and very small distances between nodes. Medium, small and very small surfaces vary between 310m * 310m to 400m* 400m, 410m * 410m to 500m* 500m, 515m * 515m to 650m * 650m respectively. Random movements are generated for each simulation. Low, medium and high mobility scenarios are generated and node speeds are limited between 1.0m/s to 5.0m/s, 5.0m/s to 10.0m/s, 10.0m/s

to 20.0m/s respectively. K heuristic of merging clustering algorithm is changed to obtain different number of clusterheads. Round-trip delay as measured against the number of clusterheads, total number of nodes, mobility and surface area are recorded. As depicted in Fig. 7, the time complexity increases linearly and at worst, the backbone formation scheme is completed in 1.5s for a MANET with 100 nodes.

For a MANET with 50 nodes, number of clusterheads are selected from 3 to 8 to measure the round-trip delay in Fig. 8. A linear increase can be seen in Fig. 8 which starts from 35ms and ends in 65ms approximately.

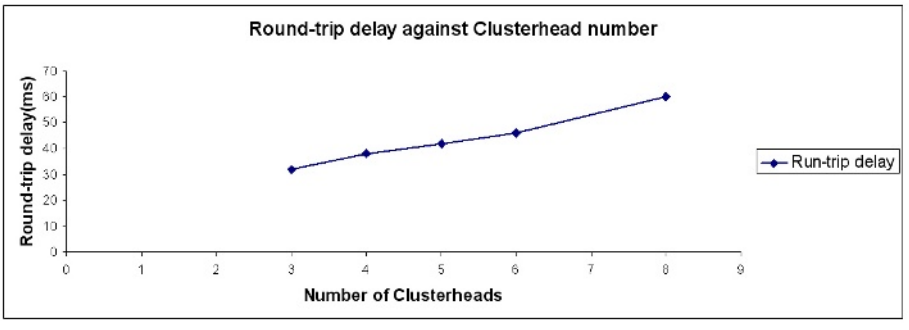


Fig. 8. Round-trip delay against number of clusterheads

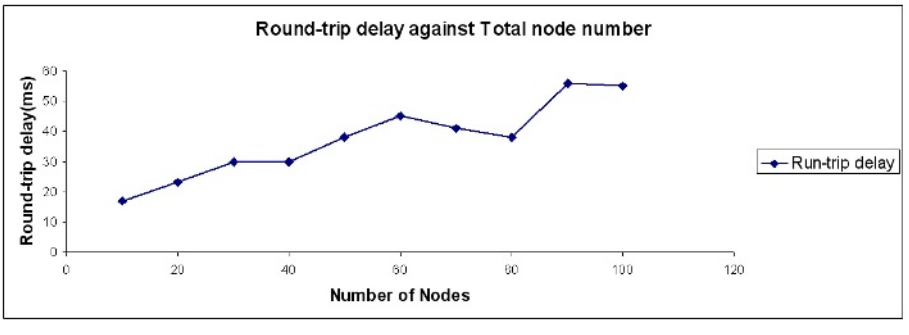


Fig. 9. Round-trip delay against number of nodes

Round-trip delay against total number of nodes is measured with constant 4 clusters and the total number of nodes are varied between 10 to 100 in Fig. 9. Round-trip delay times increase linearly from 20ms to 60ms approximately as shown in Fig. 9.

In small surface scenarios, the connectivity between nodes is higher because of small distances between the nodes and the connectivity between nodes causes a decrease in the routing delay. Fig. 10 shows the effects of distance between nodes to round-trip delay of the ring.

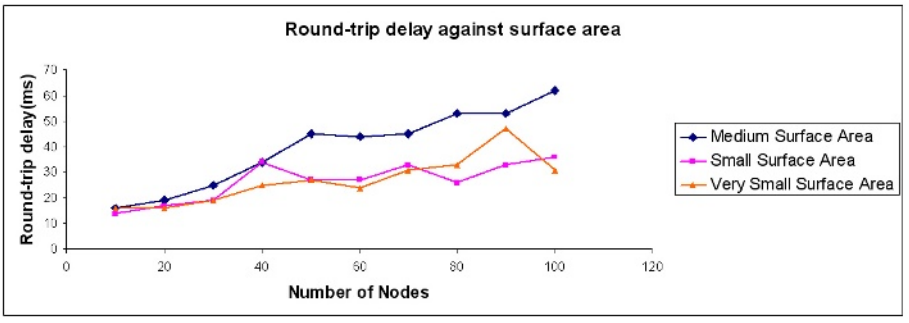


Fig. 10. Round-trip delay against surface area

Lastly, mobility parameter is changed to obtain the behavior of the algorithm with respect to mobility. Our algorithm results in approximate round-trip delay values for high mobile scenarios as shown in Fig. 11.

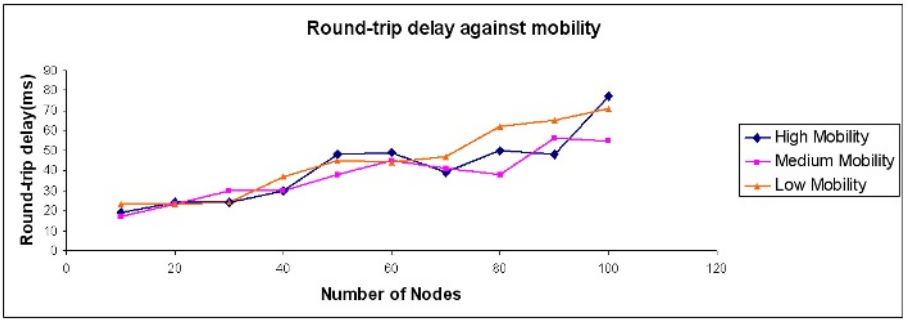


Fig. 11. Round-trip delay against Mobility

5 Conclusions

We proposed a new fully algorithm for backbone formation in MANETs and illustrated its operation. Our original idea is the construction of backbone architecture as a directed ring. The second contribution is to connect the clusterheads of a balanced clustering scheme which completes two essential needs of clustering by having balanced clusters and minimized routing delay. Besides these, the backbone formation algorithm is fault tolerant as the third contribution. The implementation results show that the algorithm is scalable in terms of its running time and round-trip delay against mobility, surface area, number of nodes and number of clusterheads. We are planning to experiment various *total order multicast* and *mutual exclusion* algorithms in such an environment where message ordering and mutual exclusion are provided by the clusterheads on behalf of the ordinary nodes of the MANET.

References

1. Yu, J.Y., Chong, P.H.J., "A survey of clustering schemes for mobile ad hoc networks", in Proc. IEEE Communications Surveys and Tutorials, 15531877, (2005).
2. Wu J., Li, H., L., "On Calculating Connected Dominating Set for Efficient Routing in Ad Hoc Wireless Networks", Proc. 3rd Intl. Wksp. Discrete Algorithms and Methods for Mobile Comp. and Commun., 714, (1999).
3. Chen Y.-Z. P., Liestman, A. L., "Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks" , in Proc. 3rd ACM Intl. Symp. Mobile Ad Hoc Net. and Comp., 165-172, (2002).
4. Cokuslu, D., Erciyes, K. and Dagdeviren, O., "A Dominating Set Based Clustering Algorithm for Mobile Ad hoc Networks", ICCS 2006, Springer Verlag, LNCS, (2006).
5. Das, B., Bharghavan, V., "Routing in Ad Hoc Networks Using Minimum Connected Dominating Sets", in Proc. IEEE ICC97, 37680, 33(2), (1997).
6. Das B., Sivakumar, R. and Bharghavan, V., "Routing in Ad Hoc Networks Using a Spine", in Proc. IEEE Intl. Comp. and Commun. Net. 97, 120, (1997).
7. Lin, C. R. and Gerla, M. "Adaptive Clustering for Mobile Wireless Networks", IEEE JSAC, 1265-1275, 15, (1997).
8. Chiang, C.-C. et al., "Routing in Clustered Multihop, Mobile Wireless Networks with Fading Channel, in Proc. IEEE SICON97, (1997).
9. Yu, J. Y., Chong, P. H. J., "3hBAC (3-hop between Adjacent Clusterheads): a Novel Non-overlapping Clustering Algorithm for Mobile Ad Hoc Networks", in Proc. IEEE Pacrim03, 31821, 1, (2003).
10. Kwon, T. J. et al.,"Efficient Flooding with Passive Clustering an Overhead-Free Selective Forward Mechanism for Ad Hoc/Sensor Networks", in Proc. IEEE, 12101220, 91(8), Aug. 2003.
11. MaDonald, A. B., Znati, T. F., "A Mobility-based Frame Work for Adaptive Clustering in Wireless Ad Hoc Networks" , IEEE JSAC, 14661487, 17, Aug. (1999).
12. Basu, P., Khan, N. and Little, T. D. C., "A Mobility Based Metric for Clustering in Mobile Ad Hoc Networks", in Proc. IEEE ICDCSW 01, 41318, Apr. (2001).
13. McDonald, A. B., Znati, T. F., "Design and Performance of a Distributed Dynamic Clustering Algorithm for Ad-Hoc Networks", in Proc. 34th Annual Simulation Symp., 2735, (2001).
14. Amis, A. D., Prakash, R., "Load-Balancing Clusters in Wireless Ad Hoc Networks", in Proc. 3rd IEEE ASSET00, 2532, (2000).
15. Wu, J. et al., "On Calculating Power-Aware Connected Dominating Sets for Efficient Routing in Ad Hoc Wireless Networks", J. Commun. and Networks, 5970, 4(1), (2002).
16. Ryu, J.-H., Song, S., Cho, D.-H., "New Clustering Schemes for Energy Conservation in Two-Tiered Mobile Ad-Hoc Networks", in Proc. IEEE ICC01, 862866, 3, (2001).
17. Dagdeviren, O., Erciyes, K., Cokuslu, D., "A Merging Clustering Algorithm for Mobile Ad hoc Networks", ICCSA 2006, Springer Verlag LNCS, (2006).
18. Ohta, T., Inoue, S. and Kakuda, Y., "An Adaptive Multihop Clustering Scheme for Highly Mobile Ad Hoc Networks", in Proc. 6th ISADS03, (2003).
19. Chatterjee, M., Das, S. K. and Turgut, D., "An On-Demand Weighted Clustering Algorithm (WCA) for Ad hoc Networks", in Proc. IEEE Globecom00, 16971701, (2000).

20. Erciyes, K., "Cluster-based Distributed Mutual Exclusion Algorithms for Mobile Networks", EUROPAR 2004, Springer-Verlag, LNCS 3149, 933-940, (2004).
21. Gallagher, R. G., Humblet, P. A., AND Spira, P. M, "A Distributed Algorithm for Minimum-Weight Spanning Trees", ACM Transactions on Programming Languages and Systems 5, 66-77, (1983).

A Service Differentiation Mechanism for Improving the Performance of IEEE 802.15.4 Sensor Networks*

Tae-Yoon Kim [1], Sungkwan Youm [2], Eui-Jik Kim [2], and Chul-Hee Kang [1]

[1] Department of Electronics Engineering, Korea University
1, 5-ga, Anam-dong, Sungbuk-gu, 136-701, Seoul, Korea
{2000kty, chkang}@widecomm.korea.ac.kr
[2] Samsung Electronics Co., LTD
416, Maetan3-dong, Paldal-gu, Suwon-si, Gyenggi-do, Korea
{sk.youm, euijik.kim}@samsung.com

Abstract. In the sensor networks, each device generates data of different sizes in the home networking and the industrial application according to their roles in the networks. In this paper, we propose a mechanism that provides differentiated services for the IEEE 802.15.4 sensor networks to improve the total throughput and the fairness of the channel. To provide differentiated services for each and every device, our mechanism adds different sizes of backoff period according to the size of packet that is generated by the device. The mathematical model based on the discrete-time Markov chain is presented and is analyzed to measure the performances of the proposed mechanism. Simulation results are also given to verify the accuracy of the analytical model. Finally, the analytical results show the improvement in the throughput and the fairness of the network which applies our mechanism.

1 Introduction

For the last few years, the researches on wireless sensor networks have been increased significantly. Terms such as pervasive computing and smart spaces are being used for describing future computing and communications. These concepts are adapted to our personal and business domains being densely populated with miniature sensors, which are constantly monitoring the environment and reporting the data to each other or to some central base stations. Sensor networks cover from small applications such as health monitoring to large applications like environment surveillance. In other words, it can be used widely in practical applications from home networking to industrial applications. The recent IEEE 802.15.4 standard for the low rate wireless personal area networks is considered as one of the technology candidates for wireless sensor networks, since it supports small, cheap, energy-efficient devices operating on battery power that require little infrastructure to operate, or none at all [1, 2].

Applications of Sensor networks are mainly used for measurement of industries and have been grown with industrial development. In this reason, requirements of

* This research was supported by the MIC (Ministry of Information and Communication), Korea, under the ITRC (Information Technology Research Center) support program supervised by the IITA (Institute of Information Technology Assessment).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 231–245, 2006.
© Springer-Verlag Berlin Heidelberg 2006

sensor networks are increased continuously. Moreover, demands for large scale sensor networks which can interact with not a few of devices are also increased because of development of instruments and machines. But there are a few problems with IEEE 802.15.4 sensor network even though many suitable aspects for sensor networks with IEEE 802.15.4 are mentioned before. IEEE 802.15.4 sensor network did not have enough channel resource or bandwidth to adapt to large scale sensor networks. And a few problems may occur due to low channel data-rate of the standard in the network which supports many devices. It is easy to exceed channel data-rate when a few number of devices that have long size packets or those with short inter-arrival rate increase in the network. For instance, in case of IEEE802.15.4 sensor network using 868-868.6MHz band, if there are equal to or more than only 5 devices that generates 512bits of data per two super-frames, total data rate of them exceeds data rate of a channel which is 20kbps. So there can be lots of collisions and performance of the network may be decreased easily.

Like previously mentioned, IEEE 802.15.4 sensor network can be used in various areas. Therefore, each device in the IEEE 802.15.4 sensor network generates different data of different sizes according to their roles in the network. In this point of view, we propose a mechanism that provides differentiated services for IEEE 802.15.4 sensor networks to improve the performance of the network by using the aspect that each device generates different size of data respectively.

We present the mathematical model for the proposed mechanisms based on IEEE 802.15.4 sensor networks, which is based on the previous works of analyzing IEEE 802.15.4 [2, 3] and IEEE 802.11[4]. We consider the beacon-enabled mode with slotted CSMA-CA algorithm in our model and assume the saturation conditions, i.e. each and every device always has packets waiting to be transmitted, for the performance analysis. The mathematical model is based on the discrete-time Markov chain in which each component of an element in state space is representing the situation of the head packet in the queue of a device. By analyzing the Markov chain, we obtain the access probability for the device and the probability that the medium is idle. Moreover, we obtain the saturation throughput and the drop probability.

2 A Survey of Legacy 802.15.4

In the beacon enabled networks, the PAN coordinator set superframes as a cycle of its channel time. Each superframe begins with the transmission of a network beacon in an active portion and an optional inactive portion. In the active portion of superframe, the coordinator interacts with its PAN and may enter a power saving mode during the inactive portion. The superframe duration, SD, is equal to the duration of the active portion of the superframe, which cannot exceed the beacon interval, BI. In the active portion, each superframe is divided into sixteen uniformly sized slots. The beacon frame is transmitted at the beginning of slot 0 which is followed by the contention access period (CAP) of the active portion. In each slot in CAP, the channel access mechanism is contention based using CSMA-CA access mechanism.

In legacy IEEE 802.15.4, the basic time unit of MAC protocol is the duration of so-called backoff period. In slotted CSMA-CA, there are channel access opportunities at the boundary of each backoff period. The actual duration of backoff period depends

on the frequency band in which 802.15.4 WPAN is operating. The standard allows the PAN to use either one of three frequency bands: 868-868.6, 902-928 and 2400-2483.5 MHz. As in the case of other contention based access control schemes, the transmission will be attempted only when the medium is idle, but withheld if the channel is busy due to packet transmission or collision.

Now we define three parameters to describe CSMA-CA protocol. NB denotes the number of times that the algorithm is required to backoff due to the unavailability of medium during channel assessment. Let CW be the contention window, i.e. the number of backoff periods that need to be clear of channel activity before the packet transmission can begin. Finally, BE denotes the backoff exponent which is related to the number of backoff periods that a device should wait before attempting to assess the channel.

When packet arrives in the queue of device, MAC sublayer of the device sets the two parameters NB and CW by zero and 2, respectively. If the device operates on battery power, BE is set to 2 or to the constant macMinBE, whichever is less. Otherwise, it is set to macMinBE (the default value of which is 3). Then the algorithm locates the boundary of next backoff period. In next step, the algorithm attempts to avoid collisions by generating random waiting time in the range of $[0, 2^{BE} - 1]$. When the waiting period is over, MAC sublayer needs to perform CW clear channel assessment (CCA) procedures, transmit the frame, and optionally wait for the acknowledgment. If the remaining time within the CAP area of the current superframe is suitably long to accommodate all of these, MAC sublayer will perform the first CCA to see whether the medium is idle. If the remaining time is not sufficient, MAC sublayer will pause until the next superframe. If the channel is busy, the values of NB and BE are increased by one (but BE cannot exceed macMaxBE, the default value of which is 5), while CW is reset to 2. If the number of retries is below or equal to macMaxCSMABackoffs (the default value of which is 5), the algorithm generates random waiting time according to current values of NB and BE, otherwise the algorithm terminates with a channel access failure status. The failure will be reported to the higher protocol layers, which can then decide whether to attempt the transmission as a new packet again or not. If the channel is idle during CCA procedures, the value of CW is decreased by one, and the channel is reviewed once more. When the value of CW becomes zero, the packet transmission may begin, provided the remaining number of backoff periods in the current superframe suffices to handle both the packet and the subsequent acknowledgment. If this is not the case, the packet transmission is postponed until the beginning of the next superframe.

3 Service Differentiation Mechanism

Every sensor node has different role in the network and generates different data according to their roles respectively. In other words, every device in the network can generate different sizes of packet. We are proposing a mechanism that divides every device into multiple groups according to their packet sizes and gives different services to every group to enhance performance of the network. Before describing the mechanism in detail, we first state several assumptions and a definition of service. We are considering the IEEE 802.15.4 sensor network model that operates in the

beacon-enable mode with slotted CSMA-CA algorithm. In this paper we only consider contention access period (CAP) and analyze the proposed mechanism in the saturation mode. When the transmitted packet collides, the packet is dropped and the device tries to transmit a new packet in the head of the queue. Packet size of each device will not be changed because the roles of devices will not change from early stage of network forming phase commonly in sensor network. We define the service differentiation as giving different amounts of channel resources to devices in the network with probability. In the following we describe the proposed mechanism in detail.

3.1 Service Differentiation Mechanism Based on Packet Size

There are plenty of collision occurrences in networks like the IEEE 802.15.4 Sensor network which have low data-rate or channel resource. So it is effective to reduce collision occurrences for enhancing performance of the networks. There are two methods which reduce collision occurrences in the networks. First method is giving differentiated services to the devices in the networks. Second method is reducing wasted time made by collisions of large packet. To satisfy both methods we make a new mechanism which is a new operation role of all devices in IEEE 802.15.4 sensor network.

All devices within the network are divided into multiple groups according to size of packet generated by them initially. Namely, devices that generate packets of similar size are gathered into a group. In the next, every group is given additional backoff periods for service differentiation based on their average packet size. The longer an average packet size is generated by the group, more number of the additional backoff period they must take. In this paper, we express the additional backoff period of the 'g' group as $T[g]$ for mathematical analysis, and all devices which are included in the same group get equal number of additional backoff period. $T[g]$ can be obtained from the average number of slots that is taken to transmit packet of the device in the 'g' group. So the size of $T[g]$ is in proportion to the average packet size of the 'g' group.

A device in 'g' group will perform the first clear channel assessment (CCA) procedure, and if a channel is sensed busy, it adds the additional backoff period as many as $T[g]$. More specifically, the device in 'g' group will initially choose a random waiting period according to backoff exponent (BE) before the CCA procedure in every backoff stage. The device performs the first CCA procedure when the waiting period is over. If the channel is sensed idle as a result of the first CCA procedure, the device performs a second CCA procedure like legacy IEEE 802.15.4 standard. However, if the channel is sensed busy after the first CCA procedure, the device adds backoff period as long as $T[g]$ unlike legacy mechanism which makes the device enter the second CCA procedure directly. When the additional waiting period becomes zero, the device enters second CCA procedure and it will transmit packet if channel is idle. However, if the channel is sensed busy during the second CCA procedure, the device will enter the next backoff stage and will generate a new random waiting time to delay the transmission of packets. The mechanism will work again equally in every backoff stage. So this mechanism enables every device to receive differentiated service accordingly as the average packet size of their groups

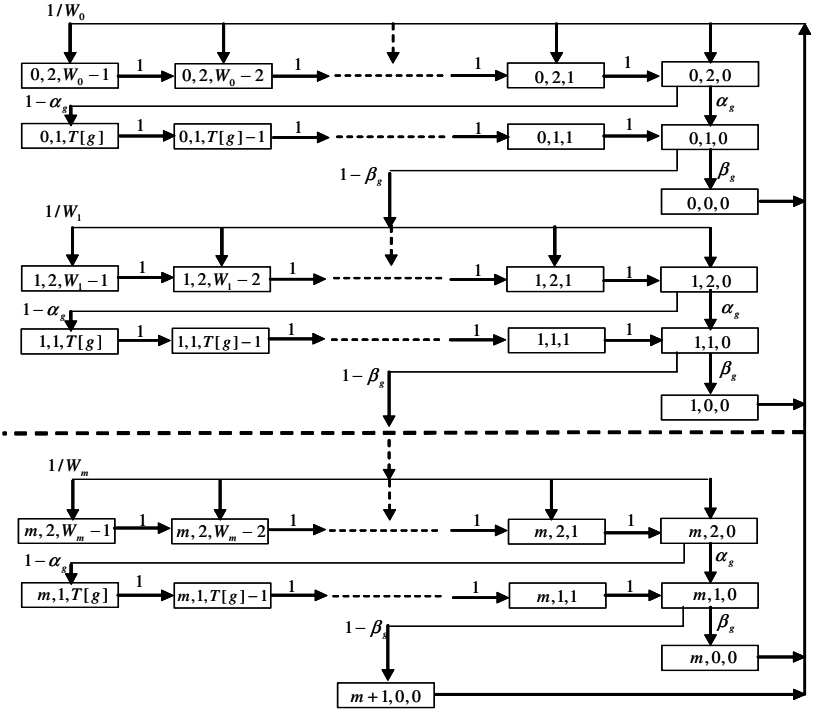


Fig. 1. Markov Chain Model

because more number of additional backoff period a device take, it have to wait more time in stochastically before transmit packets than others.

From our mechanism, it is possible to give different contention rate to each group through service differentiation. It can reduce the total collision rate of the sensor network, because the devices which receive better services will compete with the smaller number of devices than legacy mechanism that have the same contention level. Also, devices that receive worse services will yield channel access to better serviced devices and support them to access channel without a hitch. And it is also possible by using our mechanism to give lower service to the device which would like to transmit the longer sized packet. There will be fewer collisions generated during transmissions of longer size packets, which can reduce the wasted time generated by collisions. In other words, there would be more opportunities to transmit packets if the occupied time of the channel generated by collisions is reduced, which can enhance the performance of IEEE 802.15.4 sensor network.

Total throughput of the network can be enhanced by two methods we mentioned. In addition, when we defined that fairness of network is generally examined by throughput of each device, proposed mechanism also can enhance the fairness of the network. Because proposed mechanism gives more channel resource to groups which are generate packets of short size, their performance will be improved more than other groups which are generate packets of long size. In this reason, the differences between

total throughputs of groups can be decreased. More detailed explanations about the proposed mechanism are presented in following sessions with the flowchart and the Markov chain model of the mechanism.

4 Analytical Model

To analyze the proposed scheme, we introduce the following three random variables for a given device in the priority 'g' group. Let $n(g,t)$, $c(g,t)$, and $b(g,t)$ be the stochastic processes representing the value of NB, CW, and the value of the backoff period, respectively, at time t. Note that NB represents the backoff stage within the range of $[0,m+1]$, $m = macMaxCSMABackoffs$ whose default value is 4 in IEEE 802.15.4 standard. Furthermore, throughout this paper, 'g' means gth group, and gives the different priorities taking integer values in $[0,G]$, where $(G+1)$ is the number of groups with different packet size in the network. The process $\{(n(g,t),c(g,t),b(g,t))\}$ forms a multi-dimensional Markov process defining the state of the packet at the backoff unit boundaries. Since we are assuming that each device has its own priority according to their group which is not changeable, each of the processes $n(g,t)$, $c(g,t)$, $b(g,t)$ can be simplified as $n(t)$, $c(t)$, $b(t)$. $n(t)$ belongs to the range of $[0,m+1]$ in integer value. $c(t)$ may be 0, 1, or 2. Value of $b(t)$ are differed according to the value of $c(t)$.

$$b(t) = \begin{cases} 0 \sim W_i - 1, & i \in [0,m], \quad if \quad c(t) = 1 \\ T[g], & g \in [0,G], \quad if \quad c(t) = 2 \end{cases} \tag{1}$$

where $W_0 = 2^{BE}$, $W_i = 2^i W_0$.

Like previously mentioned, $T[g]$ means size of additional backoff period of group 'g'. Note that $T[g]$ is drawn by

$$T[g] = (T_H + T_{L_g} + \gamma + T_{ACK} + t_{ACK}) \times (Unit \quad backoff \quad periods/1s) \tag{2}$$

where T_H, T_{L_g}, γ, T_{ACK} and t_{ACK} denote the time to transmit the header (including MAC header, PHY header), the average time to transmit the packet of devices in group 'g', propagation delay, the time to transmit the ACK, the time to receive first bit of ACK.

The state transition diagram of these states is illustrated in Fig. 1. For the simplicity of the notations, we use the transition probabilities $P(i,j,k-1 \mid i,j,k)$ instead of $P(n(t+1)=i, c(t+1)=j, b(t+1)=k-1 \mid n(t)=i, c(t)=j, b(t)=k)$.

We assume that there is a total number of devices, n, which is composed of n_l, $l \in [0,G]$ devices in group l with their l st priority. Furthermore, we assume that each of them always has a packet ready to be transmitted. Then the one-step transition probabilities are given as follows:

$$
\begin{aligned}
P(0,2,k \mid i,0,0) &= 1/W_0, \quad i \in [0,m], \quad k \in [0,W_0-1], \\
P(0,2,k \mid m+1,0,0) &= 1/W_0, \quad k \in [0,W_0-1], \\
P(i,2,k-1 \mid i,2,k) &= 1, \quad i \in [0,m], \quad k \in [1,W_i-1], \\
P(i,1,k-1 \mid i,1,k) &= 1, \quad i \in [0,m], \quad k \in [1,T[g]], \\
P(i,1,0 \mid i,2,0) &= \alpha_g, \quad i \in [0,m], \\
P(i,0,0 \mid i,1,0) &= \beta_g, \quad i \in [0,m], \\
P(i,1,T[g] \mid i,2,0) &= 1-\alpha_g, \quad i \in [0,m], \\
P(i+1,2,k \mid i,1,0) &= (1-\beta_g)/W_{i+1}, \quad i \in [0,m-1], \quad k \in [0,W_{i+1}-1], \\
p(m+1,0,0 \mid m,1,0) &= 1-\beta_g,
\end{aligned}
\tag{3}
$$

Note that all of the states are positive recurrence and the system is stable. Therefore, there exist the stationary probabilities $\{b_{i,j,k}\}$ of the discrete-time Markov chain which can be defined as

$$
b_{i,j,k} = \lim_{t\to\infty} P\{n(t)=i, c(t)=j, b(t)=k\}, \quad i \in [0,m+1], \quad j \in [0,2] \quad \textit{and}
\tag{4}
$$

$$
k = \begin{cases} 0 \sim W_i - 1, & i \in [0,m], \quad \textit{if} \quad j=1 \\ T[g], & g \in [0,G], \quad \textit{if} \quad j=2 \end{cases}
$$

Due to the chain regularities, the following relationships hold:

$$
\begin{aligned}
b_{i,0,0} &= b_{0,0,0}(1-\beta_g)^i, \quad i \in [0,m], \\
b_{i,2,k} &= b_{0,0,0}\frac{(W_i-k)}{W_i}\frac{(1-\beta_g)^i}{\beta_g}, \quad i \in [0,m], \quad k \in [0,W_i-1], \\
b_{i,1,k} &= b_{0,0,0}\frac{(1-\alpha_g)(1-\beta_g)^i}{\beta_g}, \quad i \in [0,m], \quad k \in [1,T[g]], \\
b_{i,1,0} &= b_{0,0,0}\frac{(1-\beta_g)^i}{\beta_g}, \quad i \in [0,m], \\
b_{m+1,0,0} &= b_{0,0,0}\frac{(1-\beta_g)^{m+1}}{\beta_g},
\end{aligned}
\tag{5}
$$

Sum of all the stationary probabilities in the Markov chain will become 1. So the value of $b_{0,0,0}$ can be obtained through the following normalization:

$$
\sum_{i=0}^{m} b_{i,0,0} + \sum_{i=0}^{m}\sum_{k=0}^{W_i-1} b_{i,2,k} + \sum_{i=0}^{m}\sum_{k=1}^{T[g]} b_{i,1,k} + \sum_{i=0}^{m} b_{i,1,0} + b_{m+1,0,0} = 1
\tag{6}
$$

and subsequently we can obtain $b_{0,0,0}$ by substituting Eq. (5) into Eq. (6).

$$
\begin{aligned}
b_{0,0,0} = 2\beta_g / \{ &\sum_{i=0}^{m}(W_0 2^i + 1)(1-\beta_g)^i + 2(T[g](1-\alpha_g)+1+\beta_g)\sum_{i=0}^{m}(1-\beta_g)^i \\
&+ 2(1-\beta_g)^{m+1}\}
\end{aligned}
\tag{7}
$$

By substituting Eq. (7) into each equation in Eq. (5), we obtain the stationary probabilities $\{b_{i,j,k}\}$.

With these stationary probabilities, we find the probability that the device transmits a packet at the boundary of a backoff period which will be denoted by τ. Let τ_g be the probability that a device in the group 'g' start transmission during a generic slot time. Then we have

$$\tau_g = \sum_{i=0}^{m} b_{i,0,0} \,. \tag{8}$$

Since all groups have different priority, elements like $T[g]$, α_g and β_g are different according to group. α_g and β_g means the probabilities that the device senses channel is idle in the first and second CCA procedure respectively. And also these probabilities mean that the channel is idle at the end of backoff counting or other $(n_g - 1) + \sum_{i=0, i\neq g}^{G} n_i$ devices are not transmitting during CCA procedures of the device in the group 'g'. Assuming the average slot times that are used by devices in the network to transmit packets is T, α_g can be described by

$$\alpha_g = 1 - (1 - (1-\tau_g)^{n_g - 1} \prod_{i=0, i\neq g}^{G} (1-\tau_i)^{n_i})T, \quad g \in [0, G], \quad n = \sum_{l=0}^{G} n_l \,. \tag{9}$$

Because of β_g is depend on α_g, β_g can be obtained from Eq. (9). So β_g can be described as

$$\begin{aligned} \beta_g = {} & (1-\tau_g)^{n_g - 1} \prod_{i=0, i\neq g}^{G} (1-\tau_i)^{n_i} \\ & + 1 - (1 - (1-\tau_g)^{n_g - 1} \prod_{i=0, i\neq g}^{G} (1-\tau_i)^{n_i})T, \quad g \in [0, G], \quad n = \sum_{l=0}^{G} n_l \end{aligned} \,. \tag{10}$$

5 Performance Analysis

5.1 Throughput

Let P_I be the probability that the channel is idle because of all devices in network don't start transmission. So this probability can be calculated as followed:

$$P_I = \prod_{l=0}^{G} (1-\tau_l)^{n_l}, \quad n = \sum_{l=0}^{G} n_l \,. \,. \tag{11}$$

Let P_s and $P_{s,g}$ be the probabilities that a successful transmission occurs by a device in any priority group and a successful transmission occurs by device in the

priority 'g' group in a time slot, respectively. Then these probabilities are calculated as followed:

$$P_s = \sum_{l=0}^{G} \frac{n_l \tau_l}{1-\tau_l} \prod_{h=0}^{G} (1-\tau_h)^{n_h} = P_I \sum_{l=0}^{G} \frac{n_l \tau_l}{1-\tau_l}, \quad n = \sum_{j=0}^{G} n_j \quad and \tag{12}$$

$$P_{s,g} = n_g \tau_g (1-\tau_g)^{n_g - 1} \prod_{i=0, i \neq g}^{G} (1-\tau_i)^{n_i} = \frac{n_g \tau_g}{1-\tau_g} P_I, \quad g \in [0, G], \quad n = \sum_{j=0}^{G} n_j. \tag{13}$$

Let P_B be the probability that there is at least one transmission in the considered slot time. Then it is given by

$$P_B = 1 - P_I = 1 - \prod_{l=0}^{G} (1-\tau_l)^{n_l}, \quad n = \sum_{l=0}^{G} n_l. \tag{14}$$

Then $P_B - P_s$ is the probability that the channel is sensed busy because of the collision generated from any priority groups.

Using these equations from Eq. (11) to Eq. (14), we can calculate the normalized saturation throughput for the 'g' group, S_g.

$$\begin{aligned} S_g &= \frac{E(payload \quad transmitted \quad in \quad a \quad slot \quad time)}{E(length \quad of \quad a \quad slot \quad time)} \\ &= \frac{P_{s,g} L_g}{P_I \delta + P_s T_s + (P_B - P_s) T_c} \end{aligned} \tag{15}$$

Here T_s is the average time when the channel is busy because of a successful transmission, and T_c is the average time when the channel is busy by each station during a collision. δ is the duration of an empty slot time. L_g means average payload sizes of devices in 'g' group. T_s and T_c can be expressed by

$$T_s = T_H + T_{E(L)} + \gamma + T_{ACK} + t_{ACK} \tag{16}$$

$$T_c = T_H + T_{E(L^*)} + \gamma, \tag{17}$$

where T_H, $T_{E(L)}$, γ, T_{ACK}, t_{ACK}, L, L^* *and* $T_{E(L^*)}$ denote the time to transmit the header (including MAC header, PHY header), time to transmit average payload size of all devices in the network, propagation delay, the time to transmit the ACK, the time to receive first bit of ACK from the receiver device, payload size of each device in the network, payload size of each device in the network during a collision, the average time to transmit payload during a collision and the symbol E stands for expectations.

5.2 Packet Drop Probability

In this paper we are assuming that if a collision occurs, the packet is dropped and next packet of boundary of queue will be prepared for transmission. We are assuming this for the simplicity of analysis. Therefore, for the group 'g', the probability to be

dropped in a time slot equals to the probability that there are at least two devices which occur collision, which can be expressed as followed:

$$P_{d,g} = P_{c,g} = n_g \tau_g (1-\tau_g)^{n_g-1}(1-\prod_{l=0,l\neq g}^{G}(1-\tau_l)^{n_l}) + \sum_{k=2}^{n_g}\binom{n_g}{k}\tau_g{}^k(1-\tau_g)^{n_g-k}, \quad g \in [0,G], \quad n=\sum_{j=0}^{G} n_j, \tag{18}$$

6 Analytical and Simulation Results

In this section we compare the analytic and simulation results to verify the accuracy of the analytical model of the proposed mechanism and present the performance analysis of that mechanism. The analytic results show the effect of the proposed mechanism. Simulations are performed using a Matlab-version6.5 simulator. The the analytic results without losing the comprehensive analysis of the model. The following assumptions are applied with the saturation mode which is considered in this paper. We assume that the packet sizes of devices in each group are constant. In addition, we assume that packets for ACK are not collided.

Table 1. The parameter sets used in the analytical analysis and simulation

Packet payload	Group 1	26 bytes
	Group 2	52 bytes
	Group 3	104 bytes
Channel bit rate		20 kbits/sec
ACK		40 bits
MAC header		200 bits
macMaxCSMABackoffs		4
PHY header		48 bits
Unit backoff period		20 symbols
Modulation symbol		1 Data bit in 860MHz band

Table 2. Comparison of throughputs on the legacy 802.15.4 with varying number of devices (unit: bits/sec)

The number of devices for each group			Analysis			Simulation		
Groups			Groups			Groups		
G1	G2	G3	G1	G2	G3	G1	G2	G3
5	5	5	667.3	1,334.6	2,669.3	660.1	1,342.3	2,753.7
10	10	10	509.5	1,019.1	2,038.3	523.7	986.0	2,015.9
15	15	15	423.5	847.1	1,694.3	407.0	848.7	1,632.2
20	20	20	367.1	734.4	1,468.8	370.2	710.2	1,430.9
25	25	25	326.9	653.9	1,307.7	339.5	669.1	1,257.0

Table 3. Comparison of throughputs on the proposed mechanism with varying number of devices (unit: bits/sec)

The number of devices for each group			Analysis			Simulation		
Groups			Groups			Groups		
G1	G2	G3	G1	G2	G3	G1	G2	G3
5	5	5	667.3	1,334.6	2,669.3	660.1	1,342.3	2,753.7
10	10	10	509.5	1,019.1	2,038.3	523.7	986.0	2,015.9
15	15	15	423.5	847.1	1,694.3	407.0	848.7	1,632.2
20	20	20	367.1	734.4	1,468.8	370.2	710.2	1,430.9
25	25	25	326.9	653.9	1,307.7	339.5	669.1	1,257.0

In order to compare the network's performance of using the proposed mechanism and that of using the legacy mechanism, we made two analytic models for the both mechanisms. Analytic model of the legacy mechanism is made from the previous work of analyzing IEEE 802.15.4 [2] with a little modifying. All devices in the network are divided into multiple groups based on packet size of each device. But there is no priority for each group and all groups have the same opportunities for accessing to the channel. Analytic model for the proposed mechanism are presented in previous session. For verifying the accuracy of the analytic models, the comparisons of throughputs with a varying number of devices within each group are presented in Table 2 and Table 3. The packet size of each group is set by Table 1. As shown in the Table 2 and Table 3, the results of simulation are almost the same as those of analytic results. All simulation results in Table 2 and Table 3 are obtained with 97.89% and 98.24% Confidential Rates respectively. In the table the Confidential Rate (CR) between analytical and simulation results are given, which are calculated using the following equation:

$$CR = \left(1 - \frac{E[|S_{anal} - S_{sim}|}{E[S_{anal}]}\right) \times 100\% .$$

Since the differences between the analytical and simulation results are negligible, in the remained figures we present the analytical results only. In each figure from Fig. 2 to Fig. 5, x axis denote the number of devices at each group. We assume the number of devices in each group is all the same in the figures of analytical results. For example, if x is 5, there is the total number of 15 devices because we there are three groups in each figure with characteristics listed in Table 1. And we named the proposed mechanism as SDiPS(Service Differentiation by Packet Size).

τ in Fig. 2 represents the probabilities of trying transmission for each group characterized by the packet sizes. In the figure, y axis denote τ at each group. The value of τ is drawn by many elements of the network such as the number of devices in each group, the priority of the group, the size of backoff period of each device and so on. In brief, τ for each group in the SDiPS is different each other as we can see from Fig. 2 because the device of every group in the SDiPS has different opportunity to transmit packet by different number of additional backoff periods. We analyzed that the lowered τ values are drawn from lowered probability of sensing idle channel

during CCA procedure of the device which is one of the effects of service differentiation. And we expect that there will be more lowered τ values if there is more degree of service differentiations. SDiPS's lowered τ value means there are lower competitions for obtaining an access to the channel than the legacy mechanism and it can relief tension of the channel. Especially, the devices of groups with long packet size try a transmission with low probabilities. We think it is important factor to use channel resources efficiently.

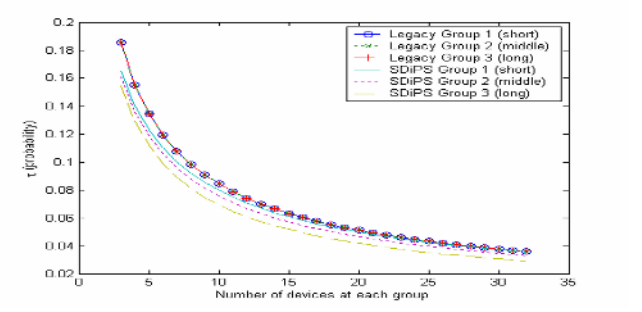


Fig. 2. τ (Probability of trying transmission)

With previous τ values, we already know that the channel access attempts are alleviated due to the service differentiation in the SDiPS. As a result of that, the collision probabilities of each group in the SDiPS shown in Fig. 3 are decreased in compared with the legacy mechanism. And the low access attempts of the group with long packet size result in fewer occurrences of collision during the transmissions of long size packets in the SDiPS. Therefore, using the SDiPS can reduce an average collision time. And it enables the devices in the network to make better use of channel resources.

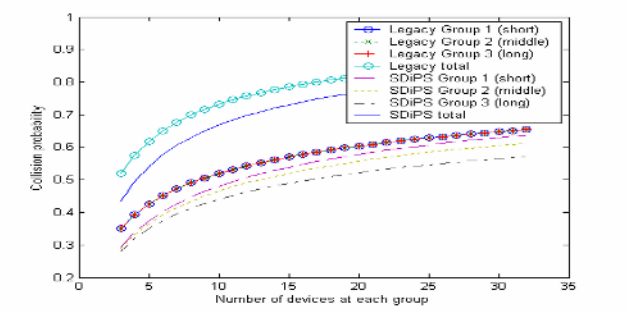


Fig. 3. Collision probability

The throughputs of each group in the legacy mechanism are different because the packet sizes are different while successful transmission probabilities of each device are the same, in Fig. 4. The total throughput of the SDiPS that sum up throughputs of each group is improved because the decrease of total collision probability presented in

Fig. 3. And the reducing collision occurrences during transmit large size packet can generate spare time to transmit more data during the same time. The throughput of the group with short packet size is improved since the group is given more opportunities by the service differentiation in the SDiPS.

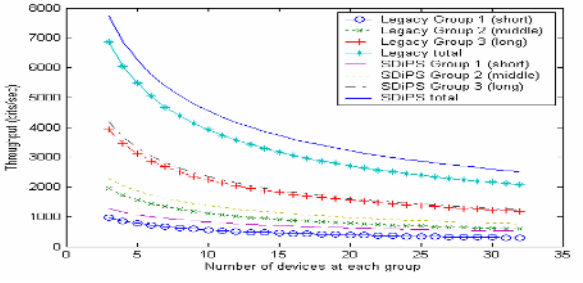


Fig. 4. Throughput

The throughput difference shows that the fairness property is improved after applying the SDiPS in Fig. 5. The throughput difference subtracts the throughput of group 3 from that of group 1. The decrease of throughput difference means that all devices share the channel resource well. And it decreases as the number of device increase, which means that using the SDiPS is more useful for the fairness when there are more devices in the channel.

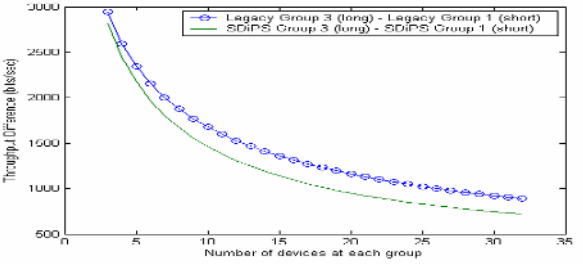


Fig. 5. Throughput Difference

Table 4. Gain between the total throughputs of the legacy mechanism and the SDiPS

The number of devices for each group			Analysis		
Group1	Group2	Group3	Proposed	Legacy	Gain
5	5	5	5,347.7	4,671.3	1.1447
10	10	10	4,123.0	3,567.0	1.1558
15	15	15	3,447.2	2,965.0	1.1626
20	20	20	3,001.7	2,570.4	1.1678
25	25	25	2,680.5	2,288.5	1.1713

Finally gain between the total throughputs of the legacy mechanism and the SDiPS are showed in Table 4. Gain is obtained from dividing SDiPS total throughput by total throughput of legacy network. As number of each device increase, the gain is also increased nearly from 14% to 17%. This means SDiPS is helpful to sensor network which interact with many devices in it. And we think that the results of this paper deserve to be considered with industrial development.

7 Conclusion

In this paper, we propose a mechanism for the IEEE 802.15.4 sensor networks which provides differentiated services to each and every device by adding different size of backoff period on each device according to the size of packet generated by the device. The mathematical model based on the discrete-time Markov chain is provided for analyzing the performance of the proposed mechanism. The comparison of analytical and simulation results are given to verify the accuracy of the numerical model. The analytical results of several performance measurements are given to analyze the effect of the proposed mechanism on the IEEE 802.15.4 sensor networks.

We could summarize the following benefits on IEEE 802.15.4 sensor network when proposed mechanism is applied to the network. First, the throughput of IEEE 802.15.4 sensor network can be improved. The collision probability can be reduced by providing contention rate differently to each device. Moreover, the wasted time generated by collision can be reduced by reducing the occurrences of collision during the transmissions of long size packets. Also, these profits of proposed mechanism are more effective as the number of devices increase. Second, the fairness property is improved because there are remarkable increases of opportunities to transmit short packets, while there are not marked increases of those to transmit long packets. Therefore the mechanism can decrease throughput differences between group of short packet and group of long packet. This means that all devices share the channel resource more equally.

However, there is a short cut of using our mechanism when all devices in network generate similar or the same sizes of packets. In that situation, our mechanism can not provide service differentiation quite well to devices in the network and there may be not much improvement. As for further work, the delay characteristics should be analyzed for each group in order to estimate how much time is taken for the devices to use the proposed mechanism to transmit packets.

References

1. Standard for part 15.4, Wireless medium access control (MAC) and physical layer (PHY) specifications for low rate wireless personal area networks (WPAN), IEEE Std 802.15.4, IEEE, New York, NY. (2003)
2. Misic, J., Shafi, S., Misic, V.B.: The Impact of MAC Parameters on the Performance of 802.15.4 PAN, Elsevier Ad hoc Networks, Vol. 2. (2004) 351-371
3. Eui-Jik Kim, Meejoung Kim, Sung-kwan Youm, Seokhoon Choi, and Chul-Hee Kang.: Priority-Based Service Differentiation Scheme for IEEE 802.15.4 Sensor Networks, will be published on Elsevier AEU, (2006)

4. Bianchi, G.: Performance Analysis of the IEEE 802.11 Distributed Coordination Function, IEEE Journal on Selected Areas in Communications, Vol. 18. (2000) 535-547
5. Robinson, J.W., Randhawa, T.S.: Saturation Throughput Analysis of IEEE 802.11e Enhanced Distributed Coordination Function, IEEE Journal on Selected Areas in Communications, Vol. 22. (2004) 917-928

Randomized Leader Election Protocols in Noisy Radio Networks with a Single Transceiver*

Jacir Luiz Bordim[1], Yasuaki Ito[2], and Koji Nakano[2]

[1] Department of Computer Science, University of Brasilia, 70910-900 Brasilia - DF, Brazil
[2] School of Engineering, Hiroshima University, Kagamiyama, Higashi-Hiroshima, 739-8527, Japan

Abstract. In this work, we present leader election protocols for single-hop, single-channel noisy radio networks that do not have collision detection (CD) capabilities. In most leader election protocols presented so far, it is assumed that every station has the ability to transmit and monitor the channel at the same time, it requires every station to be equipped with two transceivers. This assumption, however, is unrealistic for most mobile stations due to constraints in cost, size, and energy dissipation. Our main contribution is to show that it is possible to elect a leader in an anonymous radio network where each station is equipped with a single transceiver. We first present a leader election protocol for the case the number n of stations is known beforehand. The protocol runs in $O(\log f)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We then present a leader election protocol for the case where n is not known beforehand but an upper bound u of n is known. This protocol runs in $O(\log f \log u)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We also prove that these protocols are optimal. More precisely, we show that any leader election protocol elect a leader with probability at least $1 - \frac{1}{f}$ must run in $\Omega(\log f)$ time slots if n is known. Also, we proved that any leader election protocol elect a leader with probability at least $1 - \frac{1}{f}$ must run in $\Omega(\log f \log u)$ time slots if an upper bound u of n is known.

1 Introduction

In recent years, wireless and mobile communications have seen an explosive growth both in terms of the number of services provided and the types of technologies that have become available. Indeed, cellular telephony, radio paging, cellular data, and even rudimentary cellular multimedia services have become commonplace and the demand for enhanced capabilities will continue to grow into the foreseeable future [1,4,5,11,13,24]. It is anticipated that in the not-so-distant future, mobile users will be able to access their data and other services such as electronic mail, video telephony, stock market news, map services, electronic banking, while on the move [5,13,15]. In a time slot, a station can transmit or listen to the channel using a transceiver. Note that, a transceiver can

* Work supported in part by JSPS Grant-in-Aid for Scientific Research.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 246–256, 2006.
© Springer-Verlag Berlin Heidelberg 2006

perform one of the transmitting and listening operations in a time slot. Should a station need to do both operations at the same time, two transceivers are necessary. However, this assumption is unrealistic as most mobile devices are usually equipped with a single transceiver due to stringent constraints in size and power consumption.

Unlike the well-studied cellular systems that assume the existence of a robust infrastructure, radio networks must be rapidly deployable, possibly multihop, self-organizing, and capable of multimedia service support. Radio networks suit well the needs specific to disaster-relief, search-and-rescue, law-enforcement, collaborative computing, and other special-purpose applications [9,10,14,17,18].

A radio network is a distributed system with no central arbiter, consisting of n radio transceivers, henceforth referred to as *stations*. We assume that the stations are identical and cannot be distinguished by serial or manufacturing number. As customary, time is assumed to be slotted and all the stations have a local clock that keeps synchronous time, perhaps by interfacing with a GPS system. The stations are assumed to have the computing power of a usual laptop computer; in particular, they all run the same protocol and can generate random bits that provide local data on which the stations may perform computations.

We employ the commonly-accepted assumption that when two or more stations are transmitting on a channel in the same time slot, the corresponding packets *collide* and are lost. In terms of their collision detection capabilities, the radio networks come in three flavors. In the radio network with *collision detection* (CD) the status of the channel is:

NULL: if no station transmitted on the channel in the current time slot,
SINGLE: if one station transmitted on the channel in the current time slot,
COLLISION: if two or more stations transmitted in the current time slot.

In the radio network with no collision detection (no-CD) the status of a radio channel is:

NOISE: if either no station transmitted or two or more stations transmitted in the current time slot, and
SINGLE: if one station transmitted in the current time slot.

In other words, the radio network with no-CD cannot distinguish between no transmissions on the channel and the result of two or more stations transmitting at the same time. Several workers have argued that from a practical standpoint the no CD assumption makes a lot of sense since in many situations, especially in the presence of noisy channels, the stations cannot distinguish between the no transmit case and the collision of several packets that arises when several stations attempt to broadcast at once [2,3].

Note that, if a station has two transceivers, it can send a packet and can detect the status of the channel in the same time slot. However, if a station with a single transceiver sends a packet, it cannot detect the status of the channel.

The *leader election* problem asks to designate one of the stations as *leader*. In other words, after performing the leader election protocol, exactly one station

learns that it was elected leader, while the remaining stations learn the identity of the leader elected. The leader election problem is fundamental, for many protocols rely directly or indirectly, on the presence of a leader in a network [21,25]. Further, once a leader is available, the radio network with CD can be simulated by the radio network with no-CD with a constant factor slowdown [16].

It is customary to address the leader election problem on the radio network in three different scenarios:

known n (Scenario 1): Every station knows in advance the number n of stations;

known upper bound of n (Scenario 2): The upper bound u of n is known in advance. More specifically, there exists a positive integer u such that $u \leq n$ is guaranteed, and every station knows u.

unknown n (Scenario 3): The number n of stations is not known beforehand.

It is intuitively clear that the task of leader election for Scenario 1 is the easier and the hardest in Scenario 3, with Scenario 2 being in-between the two.

Several randomized protocols for single-channel radio networks have been presented in the literature. Metcalfe and Boggs [19] presented a simple leader election protocol for the radio network with no-CD for known n that is guaranteed to terminate in $O(1)$ expected rounds. For unknown n, several protocols have been proposed for the radio network with CD and no-CD. Willard [25] showed that the leader election on the radio network with CD can be solved in $\log\log n + o(\log\log n)$ expected time slots. Later, Nakano and Olariu [21] presented two leader election protocols for the radio network with CD that terminate in $O(\log n)$ time slots with probability at least $1 - \frac{1}{n}$ and in $O(\log\log n)$ time slots with probability at least $1 - \frac{1}{\log n}$. Recently, Nakano and Olariu [22] improved the protocol of [21] showing that the leader election on the radio network with CD can be performed in $\log\log n + 2.78\log f + o(\log\log n + \log f)$ time slots with probability at least $1 - \frac{1}{f}$ for every $f > 1$. Hayashi *et al.* [16] proposed a leader election protocol for the radio network with no-CD that terminates in $O((\log n)^2)$ time slots with probability at least $1 - \frac{1}{n}$. Nakano and Olariu [23] have presented that a leader can be elected in $O(\log f)$ time slots with probability at least $1 - \frac{1}{f}$ for every $f > 1$ if every station knows the number of stations.

All of the above protocols assumes that every station is equipped with two transceivers, and transmit and monitor the channel at the same time. This assumption, however, is unrealistic for most mobile stations due to constraints in cost, size, and energy dissipation. Quite recently, we have shown that, even if every station is equipped with a single transceiver, a leader can be elected in $\log\log n + o(\log\log n) + O(\log f)$ time slots with probability at least $1 - \frac{1}{f}$ for every $f > 1$ in the radio network with collision detection capabilities (CD) [6] .

Our main contribution is to show that it is possible to elect a leader in an anonymous radio network with no collision detection capabilities (no-CD) where

each station is equipped with a single transceiver. We first present a leader election protocol for the case the number n of stations is known beforehand. The protocol runs in $O(\log f)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We then present a leader election protocol for the case where n is not known beforehand but an upper bound u of n is known. This protocol runs in $O(\log f \log u)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We prove that these protocol are optimal. More precisely, we show that any leader election protocol elect a leader with probability at least $1 - \frac{1}{f}$ must run in $\Omega(\log f)$ time slots if n is known. Also, we have proved that any leader election protocol elect a leader with probability at least $1 - \frac{1}{f}$ must run in $\Omega(\log f \log u)$ time slots if an upper bound u of n is known.

2 A Refresher of Basic Probability Theory

This section offers a quick review of basic probability theory results that are useful for analyzing the performance of our randomized leader election protocols. For a more detailed discussion of background material we refer the reader to [20].

Throughout, $\Pr[A]$ will denote the probability of event A. For a random variable X, $E[X]$ denotes the expected value of X. Let X be a random variable denoting the number of successes in n independent Bernoulli trials with parameters p and $1 - p$. It is well known that X has a *binomial distribution* and that for every r, $(0 \leq r \leq n)$,

$$\Pr[X = r] \;=\; \binom{n}{r} p^r (1-p)^{n-r}.$$

Further, the expected value of X is given by

$$E[X] = \sum_{r=0}^{n} r \cdot \Pr[X = r] = np.$$

For all $n \geq 2$, we have the inequality

$$\frac{1}{4} \leq (1 - \frac{1}{n})^n < \frac{1}{e} < (1 - \frac{1}{n})^{n-1} \leq \frac{1}{2},$$

where $e = 2.71828\cdots$ is the base of the natural logarithm. For later reference, we state the following result.

Lemma 1. *Let X be a random variable taking on a value smaller than or equal to $x(f)$ with probability at most f, $(0 \leq f \leq 1)$, where x is a non-decreasing function. Then, $E[X] \leq \int_0^1 x(f)df$.*

3 Randomized Leader Election for Known n (Scenario 1)

The main goal of this section is to provide leader election protocols for radio networks where the number n of stations in known beforehand.

Let U be a set of all stations. We assume that U has at least two stations, that is, $|U| = n \geq 2$. If U has a single station, the unique station can be elected as a leader immediately without any broadcast and computation.

Let A and B be disjoint subsets of U, that is, $A \subseteq U$, $B \subseteq U$, $A \cap B = \emptyset$, and $|U| = n$ holds. Also, let $C = U - A - B$ be the complement of $A \cup B$. The following protocol `Leader_Election(`A`,`B`)` finds a leader in three time slots if $|A| = |B| = 1$ and a single station station in A is declared as a leader.

Protocol `Leader_Election(`A`,`B`)`
Time Slot 1: Every station in A broadcasts on the channel. Stations in B and C monitor the channel.
Time Slot 2: Every station in B broadcasts on the channel if the status of the channel at time slot 1 is SINGLE. Stations in A and C monitor the channel.
Time Slot 3: Every station in A broadcasts on the channel if the status of the channel at time slot 2 is SINGLE. Stations in B and C monitor the channel.

Clearly, if $|A| = 1$ and $|B| = 1$ then the status of the channel in both time slots 2 and 3 is SINGLE. Otherwise, that is, $|A| \neq 1$ or $|B| \neq 1$ then the status of the channel in these time slots is NOISE. Thus, if the status of the channel in time slot 2 is SINGLE, a single station in A declared as a leader and stations in C learn that a leader is elected and they are not leader. If the status of the channel in time slot 3 is SINGLE the unique station in B learns that a leader has been elected.

The readers may think that the first time slots are sufficient to elect a leader and time slot 3 is not necessary. Note that all stations in U need to know if the leader has been elected. Thus, we need time slot 3 to let stations in B learn the identity of the leader elected.

The following protocol `Election(`n`)` elects a leader.

Protocol `Election(`n`)`
Step 1 Every station flips a fair coin and belongs to A with probability $\frac{1}{n}$.
Step 2 Every station in $U - A$ flips a fair coin and belongs to B with probability $\frac{1}{n-1}$.
Step 3 Execute `Leader_Election(`A`,`B`)`.

Steps 1 and 2 need no broadcast time slots, and Step 3 uses three time slots. Thus, `Randomized_Election(`n`)` runs in three time slots. Also, we can prove that $|A| = |B| = 1$ with probability at least $\frac{1}{e^2}$ as follows. Since $|A|$ follows the n independent Bernoulli trials with parameter $\frac{1}{n}$, from (1) the probability that $|A| = 1$ is

$$\Pr[|A| = 1] = \binom{n}{1} \frac{1}{n} (1 - \frac{1}{n})^{n-1} = (1 - \frac{1}{n})^{n-1} > \frac{1}{e}.$$

Suppose that $|A| = 1$. Similarly, the probability that $|B| = 1$ is

$$\Pr[|A| = 1 \mid |B| = 1] = \binom{n-1}{1} \frac{1}{n-1} (1 - \frac{1}{n-1})^{n-2} = (1 - \frac{1}{n-1})^{n-2} > \frac{1}{e}.$$

Thus, the probability that $|A| = |B| = 1$ is

$$\Pr[|A| = |B| = 1] = \Pr[|A| = 1] \cdot \Pr[|A| = 1 \mid |B| = 1] > \frac{1}{e^2}.$$

Therefore, a single trial of `Election(`n`)` elects a leader with probability at least $\frac{1}{e^2}$.

Suppose that `Election(`n`)` is repeated until $|A| = |B| = 1$ and a leader is elected. We will evaluate the number of time slots spent to elect a leader. Suppose that `Election(`n`)` are repeated t times. All of the t executions of `Election(`n`)` fail to elect a leader leader is at most $(1 - \frac{1}{e^2})^t$. It follows that with probability exceeding $1 - (1 - \frac{1}{e^2})^t$ the protocol elects a leader in at most t time slots. Note that $1 - \frac{1}{e^2} = 0.86466\cdots$. Let f be a real number satisfying $\frac{1}{f} = (1 - \frac{1}{e^2})^t$. Then, $t = O(\log f)$ holds. Hence, the protocol terminates, with probability exceeding $1 - \frac{1}{f}$, in $O(\ln f)$ time slots. Thus, we have the following result.

Theorem 1. `Election(`n`)` *succeeds in electing a leader with probability at least* $\frac{1}{e^2}$. *Also, by repeating* `Election(`n`)` *a leader can be elected in* $O(\log f)$ *time slots, with probability at least* $1 - \frac{1}{f}$ *for any* $f > 1$.

From Lemma 1, the expected running time slots of `Election(`n`)` is $\int_0^1 \log f df = O(1)$.

We also prove the optimality of `Election(`n`)`. To complete the leader election, the status of the channel must be SINGLE in at least one time slot. Let U ($|U| = n \geq 2$) be a set of all stations. Suppose that every station broadcast with probability p in the first time slot. Let X be the random variable denoting the number of stations that broadcast to the channel. Then, the status of the channel is SINGLE with probability

$$\Pr[X = 1] = \binom{n}{1} p(1-p)^{n-1} = np(1-p)^{n-1}.$$

The derivative of $\Pr[X = 1]$ for p is

$$\frac{d\Pr[X = 1]}{dp} = n(1-p)^{n-1} - np(1-p)^{n-2} = n(1-np)(1-p)^{n-2}$$

We have $\frac{d\Pr[X=1]}{dp} = 0$ when $np = 1$. Thus, $\Pr[X = 1]$ is the maximum when $np = 1$ and $\Pr[X = 1] \leq \frac{1}{2}$ for every n (≥ 2) and p ($0 \leq p \leq 1$). The equality holds when $n = 2$ and $p = \frac{1}{2}$. Therefore, the status of the channel is SINGLE with probability no more than $\frac{1}{2}$. We will show that, for any leader election protocol in the radio network with no CD, the status of the channel is SINGLE with probability at most $\frac{1}{2}$ in every time slot.

In the leader election protocol, every station can have a history which is represented by a sequence of bits, it can have no other information. At the beginning of the k-th time slot, every station has a history of $k - 1$ bits such that, if it broadcast in j-th ($1 \leq j \leq k-1$) time slot, the j-th bit is 1, and if it did not broadcast, the j-th bit is 0. Then, the leader election protocol can be simply represented by

a function $p : \{0,1\}^* \rightarrow [0,1]$, where $\{0,1\}^*$ denotes the set of all bits of length at least 0 and $[0,1]$ denotes a set of all real numbers from 0 to 1. For example, if a station has broadcast at time slot 1 and has not broadcast at time slots 2 and 3, then it broadcasts at time slot 4 with probability $p(100)$. Let $q : \{0,1\}^* \rightarrow [0,1]$ be the function such that a station has a history h with probability $q(h)$. Clearly, $q(\epsilon) = 1$, $q(0) = p(0)$, $q(1) = p(1)$, $q(00) = q(0) \cdot (1 - p(0))$, $q(01) = q(0) \cdot p(0)$, where ϵ denotes a sequence of bits with length 0. In general, for every $h \in \{0,1\}^*$ and $x \in \{0,1\}$,

$$q(h0) = q(h) \cdot (1 - p(h))$$
$$q(h1) = q(h) \cdot p(h)$$

holds. The probability $s(k)$ that a particular station broadcasts at time slot k is

$$s(k) = \sum_{h \in \{0,1\}^{k-1}} q(h) \cdot p(h).$$

Clearly, for all $k \geq 1, 0 \leq s(k) \leq 1$ holds. Let X_k be the random variable denoting the number of stations that broadcast at time slot k. Then, the probability that the status of the channel at time slot k is SINGLE is

$$\Pr[X_k = 1] = \binom{n}{1} s(k)(1 - s(k))^{n-1} \leq \frac{1}{2}.$$

Therefore, the status of the channel is SINGLE with probability at most $\frac{1}{2}$ for every time slot until the leader is elected. Thus, the leader election protocol represented by p runs in t time slots with probability at least $\frac{1}{2^t}$. Thus, we have,

Theorem 2. *Any leader election protocol that elects a leader with probability at least* $1 - \frac{1}{f}$ *need to run in* $\Omega(\log f)$ *time slots.*

From Theorem 2, the leader election protocol for Theorem 1 is optimal.

4 Randomized Leader Election for Known Upper Bound u (Scenario 2)

The main purpose of this section is to develop a randomized leader election protocol for an n-station radio network under the assumption that an upper bound u of the number n of stations is known beforehand. However, the actual value of n is not known. We assume that $n \geq 2$, because if $n = 1$, the leader election is not possible.

Let $n_1, n_2, \ldots, n_k$ be a sequence of positive numbers. The following protocol `Election`$(n_1, n_2, \ldots, n_k)$ is a generalization of `Election(`n`)`.

Protocol `Election`$(n_1, n_2, \ldots, n_k)$
for $i = 1$ **to** k **do**
  begin
    Execute Steps 1 to 3 of `Election`(n_i);
    Terminate the protocol **if** the leader is elected;
  end

For simplicity, we assume that the upper bound u is a power of two. If u is not a power of two, we can choose a minimum u' such that $u' > u$ and u' is a power of two. Clearly, such u' is an upper bound of n. Our randomized leader election protocol for Scenario 2 simply executes `Election`$(2^1, 2^2, \ldots, 2^{\log u})$.

Let us evaluate the probability that `Election`$(2^1, 2^2, \ldots, 2^{\log u})$ succeeds in electing a leader. Let i be an integer such that $2^{i-1} < n \leq 2^i$. Suppose that `Leader_Election`(2^i) is executed. The probability that $|A| = 1$ is

$$\begin{aligned}\Pr[|A| = 1] &= \binom{n}{1} \frac{1}{2^i}(1 - \frac{1}{2^i})^{n-1} \\ &= \frac{n}{2^i}((1 - \frac{1}{2^i})^{2^i-1})^{\frac{n-1}{2^i-1}} \\ &> \frac{1}{2e} \quad (\text{from } 2^{i-1} < n \text{ and } n - 1 \leq 2^i - 1).\end{aligned}$$

Similarly, we can prove

$$\Pr[|A| = 1 \mid |B| = 1] > \frac{1}{2e}.$$

in the same manner. Thus, the probability that $|A| = |B| = 1$ is

$$\Pr[|A| = |B| = 1] = \Pr[|A| = 1] \Pr[|A| = 1 \mid |B| = 1] > \frac{1}{4e^2}.$$

Therefore, a single trial of `Election`(2^i) elects a leader with probability at least $\frac{1}{4e^2}$ provided that $2^{i-1} < n \leq 2^i$. For every n such that $2 \leq n \leq u$, there exists an integer i $(1 \leq i \leq \log u)$ such that $2^{i-1} < n \leq 2^i$. Therefore, we have,

Lemma 2. *For every n such that $2 \leq n \leq u$, protocol* `Election`$(2^1, 2^2, \ldots, 2^{\log u})$ *succeeds in electing a leader in $O(\log u)$ time slots with probability at least $\frac{1}{4e^2}$.*

We further generalize `Election` for infinite sequences. Let $n_1, n_2, \ldots$ be an infinite sequence. Protocol `Election`$(n_1, n_2, \ldots)$ is defined as follows:

Protocol `Election`$(n_1, n_2, \ldots)$
for $i = 1$ **to** ∞ **do**
  begin
    Execute Steps 1 to 3 of `Election`(n_i);
    Terminate the protocol **if** the leader is elected;
  end

Let $D^1_{\log u}$ be a sequence $2^1, 2^2, \ldots, 2^{\log u}$, and $D^{k+1}_{\log u} = D^k_{\log u} \cdot D^1_{\log u}$ for all $k \geq 1$, where "$\cdot$" denotes the operator of concatenation of two sequences. Clearly, $D^k_{\log u}$ has $k \log u$ integers. Also, let $D^\infty_{\log u}$ be the infinite sequence $D^1_{\log u} \cdot D^1_{\log u} \cdots$. We have proved that in Lemma 2, `Election`$(D^1_{\log u})$ elects a leader in three time slots with probability $\frac{1}{4e^2}$. Thus, `Election`$(D^k_{\log u})$ fails to elect a leader in no more than $3k \log u$ time slots with probability at most $(1 - \frac{1}{4e^2})^k$. Let $\frac{1}{f} = (1 - \frac{1}{4e^2})^k$. Then, $3k \log u = O(\log f \log u)$. Therefore, we have,

Theorem 3. *Protocol* `Election`$(D^{\infty}_{\log u})$ *elects a leader in* $O(\log f \log u)$ *time slots with probability at least* $1 - \frac{1}{f}$ *for any* $f > 1$.

From Lemma 1, the expected running time slots of `Election`(n) is $\int_0^1 \log f \log u df = O(\log u)$.

We will show that, any leader election protocol for Scenario 2 need to run in $\Omega(\log u)$ time slots to elect a leader with probability at least $\frac{1}{2}$.

Let $X(n,p)$ denote the random variable denoting the number of stations that have broadcast if each of the n stations broadcast with probability p. The probability that the status of the channel is SINGLE with probability

$$\Pr[X(n,p)=1] = \binom{n}{1} p^1(1-p)^{n-1} = np(1-p)^{n-1}.$$

Let us evaluate the upper bounds of $\Pr[X(2^1,p) = 1]$,$\Pr[X(2^2,p) = 1]$, ..., $\Pr[X(2^{\log u},p) = 1]$ and then compute $\Pr[X(2^1,p) = 1] + \Pr[X(2^2,p) = 1] + \cdots + \Pr[X(2^{\log u},p) = 1]$. Let $m = \frac{1}{p}$. For simplicity, we assume m is an integer and a power of two. It is easy to show the upper bounds when this is not the case. Since $\Pr[X(n,p) = 1] = np(1-p)^{n-1} \leq np$, we have

$$\begin{aligned}&\Pr[X(2^1,p)=1] + \Pr[X(2^2,p)=1] + \cdots + \Pr[X(2^{\log m-1},p)=1]\\ &= 2^1p + 2^2p + \cdots + 2^{\log m-1}p = 2^1/m + 2^2/m + \cdots + 2^{\log m-1}/m < 1.\end{aligned}$$

If $n = m$ then,

$$\Pr[X(2^{\log m},p)=1] = mp(1-p)^{m-1} = (1-\frac{1}{m})^{m-1} < \frac{1}{2}.$$

Also, for every $n > m$, we have

$$\begin{aligned}&\Pr[X(2^{\log m+1},p)=1] + \Pr[X(2^{\log m+2},p)=1] + \Pr[X(2^{\log m+3},p)=1] + \cdots\\ &= \Pr[X(2^1m,p)=1] + \Pr[X(2^2m,p)=1] + \Pr[X(2^3m,p)=1] + \cdots\\ &= 2^1mp(1-p)^{2^1m-1} + 2^2mp(1-p)^{2^2m-1} + +2^3mp(1-p)^{2^3m-1} + \cdots\\ &= mp(1-p)^{m-1}(2^1(1-p)^{2^1m-m} + 2^2(1-p)^{2^2m-m} + 2^3(1-p)^{2^3m-m} + \cdots)\\ &< \frac{1}{2}(2^1(\frac{1}{e})^{2^1-1} + 2^2(\frac{1}{e})^{2^2-1} + 2^3(\frac{1}{e})^{2^3-1} + \cdots) < 1.\end{aligned}$$

Therefore, $\Pr[X(2^1,p)=1] + \Pr[X(2^2,p)=1] + \cdots + \Pr[X(2^{\log u},p)=1] < \frac{5}{2}$.

Let $s(k)$ be the probability that a particular station broadcast at time slot k. To elect a leader with probability at least $\frac{1}{2}$ in t time slots for all $n = 2^1, 2^2, \ldots, 2^{\log u}$,

$$\sum_{k=1}^{t} \Pr[X(2^1,s(k))=1] + \Pr[X(2^2,s(k))=1] + \cdots + \Pr[X(2^{\log u},s(k))=1] \geq \frac{\log u}{2}$$

must hold. However, the left hand side of the inequality is at most $\frac{5}{2}t$. Therefore, since $\frac{5}{2}t \geq \frac{\log u}{2}$, we have $t = \Omega(\log u)$. Hence, to elect a leader with probability $1 - \frac{1}{f}$ for all $n = 2^1, 2^2, \ldots, 2^{\log u}$, any leader election protocol need to run in $\Omega(\log f \log u)$ time slots.

Theorem 4. *Any leader election protocol runs that elects a leader with probability at least* $1 - \frac{1}{f}$ *need to run in* $\Omega(\log f \log u)$ *time slots.*

Therefore, protocol for Theorem 3 is optimal.

5 Conclusions

In this work, we have presented leader election protocols for single-hop, single-channel noisy radio networks that do not have collision detection (CD) capabilities. Also, we have assumed that every station is equipped with a single transceiver. We presented a leader election protocol for the case the number n of stations is known beforehand. that runs in $O(\log f)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We then presented a leader election protocol for the case where n is not known beforehand but an upper bound u of n is known. This protocol runs in $O(\log f \log u)$ time slots with probability at least $1 - \frac{1}{f}$ for any $f > 1$. We also proved that these leader elections are optimal.

References

1. D. J. Baker, Data/voice communication over a multihop, mobile, high frequency network, *Proc. MILCOM'97*, Monterey, CA, 1997, 339–343.
2. R. Bar-Yehuda, O. Goldreich, and A. Itai, Efficient emulation of single-hop radio network with collision detection on multi-hop radio network with no collision detection, *Distributed Computing*, 5, (1991), 67–71.
3. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization, *Journal of Computer and Systems Sciences*, 45, (1992), 104–126.
4. D. Bertzekas and R. Gallager, *Data Networks*, Second Edition, Prentice-Hall, 1992.
5. U. Black, *Mobile and Wireless Networks*, Prentice-Hall, Upper Saddle River, NJ, 1996.
6. J. L. Bordim, Y. Ito, and K. Nakano, An Energy Efficient Leader Election Protocol for Radio Network with a Single Transceiver, *IEICE Trans. on Fundamentals*, E89-A, 5, (2006), 1355–1361.
7. I. Chlamtac, C. Petrioli, and J. Redi, Energy-conserving access protocols for ID-NETs, Technical Report 96-006, Boston University, March 1996.
8. A. Chockalingam and M. Zorzi, Energy efficiency of media access protocols for mobile data networks, *IEEE Transactions on Communications*, COM-46, (1998), 1418–1421.
9. B. H. Davies and T. R. Davies, Applications of packet switching technology to combat net radio, *Proceedings of the IEEE*, 75, (1987), 43–55.
10. D. Duchamp, S. K. Feiner, and G. Q. Maguire, Software technology for wireless mobile computing, *IEEE Network Magazine*, Nov. 1991, 12–18.
11. K. Feher, *Wireless Digital Communications*, Prentice-Hall, Upper Saddle River, NJ, 1995.
12. W. C. Fifer and F. J. Bruno, Low cost packet radio, *Proceedings of the IEEE*, 75, (1987), 33–42.
13. V. K. Garg and J. E. Wilkes, *Wireless and Personal Communication Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1996.

14. M. Gerla and T.-C. Tsai, Multicluster, mobile, multimedia radio network, *Wireless Networks*, 1, (1995), 255–265.
15. E. P. Harris and K. W. Warren, Low power technologies: a system perspective, *Proc. 3-rd International Workshop on Multimedia Communications*, Princeton, 1996.
16. T. Hayashi, K. Nakano, and S. Olariu, Randomized initialization protocols for Packet Radio Networks, *Proc. 13th International Parallel Processing Symposium*, (1999), 544–548.
17. C. R. Lin and M. Gerla, Adaptive clustering in mobile wireless networks, *IEEE Journal on Selected Areas in Communications*, 16, (1997), 1265–1275.
18. W. Mangione-Smith and P. S. Ghang, A low power medium access control protocol for portable multimedia devices, *Proc. Third International Workshop on Mobile Multimedia Communications*, Princeton, NJ, September 1996.
19. R. M. Metcalfe and D. R. Boggs, Ethernet: distributed packet switching for local computer networks, *Communications of the ACM*, 19, (1976), 395–404.
20. R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
21. K. Nakano and S. Olariu, Randomized $O(\log \log n)$-round leader election protocols in Packet Radio Networks, *Proc. of International Symposium on Algorithms and Computation*, (LNCS 1533), 209–218, 1998.
22. K. Nakano and S. Olariu, Randomized Leader Election Protocols for Ad-hoc Networks *Proc. of 7th International Colloquium on Structural Information and Communication Complexity* (Scirocco), pp. 253–267, (2000).
23. K. Nakano and S. Olariu, Randomized Leader Election Protocols in Radio Networks with no Collision Detection, *Proc. of International Symposium on Algorithms and Computation*, 362–373, (2000).
24. C.-K. Toh *Ad Hoc Mobile Wireless Networks: Protocols and Systems*, Prentice Hall, 2002.
25. D. E. Willard, Log-logarithmic selection resolution protocols in a multiple access channel, *SIAM Journal on Computing*, 15, (1986), 468–477.

A Hybrid Intelligent Preventive Fault-Tolerant QoS Unicast Routing Scheme in IP over DWDM Optical Internet*

Xingwei Wang, Shuxiang Cai, Nan Gao, and Min Huang

College of Information Science and Engineering, Northeastern University, Shenyang, 110004, P.R. China
wangxw@mail.neu.edu.cn

Abstract. IP over DWDM (Dense Wavelength Division Multiplexing) optical Internet is one of the mainstream networking techniques for NGI (Next Generation Internet) backbone, and how to improve its fault-tolerance capability and QoS provision becomes critical. Fault-tolerant QoS routing is one of the effective solutions. In this paper, a preventive fault-tolerant QoS unicast routing scheme is proposed based on a hybrid intelligent algorithm, taking advantage of both ant-colony algorithm and genetic algorithm. Simulation results have shown that the proposed scheme is both feasible and effective with better performance.

1 Introductions

IP over DWDM (Dense Wavelength Division Multiplexing) optical Internet is one of the mainstream networking techniques for NGI (Next Generation Internet) [1-2]. How to avoid service interruption and thus make its corresponding loss lowest becomes critical. Fault-tolerance has already become one of the essential capabilities of the IP over DWDM optical Internet [3-4]. Meanwhile, the popularity of the networked multimedia applications requires the NGI to support QoS (Quality of Service). Thus, it is necessary to provide fault-tolerant QoS routing support in IP over DWDM optical Internet [3-4].

According to whether the necessary resource is reserved before failure occurred or it is allocated dynamically after failure happened, the corresponding fault-tolerant QoS routing can be classified into preventive one and reactive one [4-5]. In the former, the primary route is established at the same time the backup one is assigned with the necessary amount of resource reserved, so that the network could recover from failure in time. However, it is inefficient in terms of resource utilization, although its restoration time is relatively short. In the latter, the backup route is not assigned and thus no corresponding resource is reserved while the primary one is established. In case of the primary route failed,

* This work is supported by the National Natural Science Foundation of China under Grant No. 60673159; Program for New Century Excellent Talents in University; Specialized Research Fund for the Doctoral Program of Higher Education; the Natural Science Foundation of Liaoning Province under Grant No. 20062022.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 257–266, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the necessary resource is looked for at that time to reestablish it dynamically. It has high bandwidth utilization with long restoration time. In addition, it cannot guarantee recovering from failure in time successfully due to no resource reservation in advance.

It has been proven that the problem of finding a path subject to constraints on two or more additive or multiplicative metrics in any possible combination is NP-complete [6]. Thus, it should adopt heuristic algorithm or intelligent optimization algorithm for such kind of QoS routing problem. In this paper, based on preventive fault-tolerant strategy and primary-backup multiplexing mechanism [7], a preventive fault-tolerant QoS unicast routing scheme is proposed based on a hybrid intelligent algorithm, taking advantage of both ant-colony algorithm and genetic algorithm [8-10]. Simulation results have shown that the proposed scheme is both feasible and effective with better performance when solving the fault-tolerant QoS unicast routing problem in IP over DWDM optical Internet.

2 Model Description

2.1 Problem Description

First of all, some used terms in this paper are introduced as follows.

Link: it refers to a segment of directed optical fiber which connects two nodes in the network.

Channel: it refers to a wavelength along a link.

Path: it is composed of a set of links from the source node to the destination node.

Light-path: it is composed of a set of channels from the source node to the destination node.

Connection: it refers to the specific primary light-path and its backup one setup for one QoS light-path establishment request.

Orphan Connection: it is caused by the primary-backup multiplexing mechanism adopted in the proposed scheme in this paper. If any channel of one backup light-path in a connection is shared by another primary light-path in other connection, call it orphan connection.

In this paper, the problem to be solved is as follows: establish the primary and backup QoS-constrained light-path pair simultaneously for the specific QoS light-path establishment request with the objective of minimizing the number of orphan connections in the network.

2.2 Network Model

IP over DWDM optical Internet can be modeled by a graph $G(V, E)$, where V is the set of nodes (wavelength router or OXC) and E is the set of edges (optical fiber). Each edge is composed of two reversed optical fibers. Each directed optical fiber is called as a link. Assume the number of wavelengths that each link can support is W, and only some nodes are equipped with full-range wavelength

converter [11]. When the nodes have wavelength conversion capabilities and the wavelength conversion is necessary, the corresponding wavelength conversion operations can be done on their received or split signals. Suppose the conversion between any two wavelengths has the same conversion delay.

For each link $e_{ij} = (v_i, v_j) \in E$, $i, j = 1, 2, ..., |V|$, the following parameters are considered: set about wavelength usage description, $\Lambda(e_{ij}) = \{Q_1, Q_2, ..., Q_w\}$, Q_x is the set of light-paths which occupy the xth wavelength channel through the link e_{ij}, $x = 1, 2, ..., W$; delay $d(e_{ij})$, $d(e_{ij}) = d(e_{ji})$; link failure rate $l(e_{ij})$.

For each node $v_i \in V$, $i = 1, 2, ..., |V|$, the following parameters are considered: wavelength conversion indicator, $s(v_i)$, and when the node v_i has wavelength conversion capability, $s(v_i) \equiv 1$, otherwise $s(v_i) \equiv 0$; wavelength conversion delay, $t(v_i)$, and when the node v_i has wavelength conversion capability and does conversion, $t(v_i) \equiv t$, otherwise $t(v_i) \equiv 0$; delay jitter $dj(v_i)$.

2.3 Mathematical Model

First of all, some symbols are defined as follows:

R=$\{R_{sd} | R_{sd}$ is the QoS light-path establishment request from v_s to v_d, $\forall v_s$, $v_d \in V\}$: set of the QoS light-path establishment requests, where v_s is the source node of the R_{sd} and v_d is the destination node of the R_{sd};

Op :number of the orphan connections in the network.

$la_{ij\lambda}$: if the wavelength λ is free along the link e_{ij}, $la_{ij\lambda} = 1$, otherwise $la_{ij\lambda} = 0$, $\forall e_{ij} \in E$.

$lb_{ij\lambda}^{R_{sd}}$: if the wavelength λ along the link e_{ij} has already been allocated as the backup channel for R_{sd}, $lb_{ij\lambda}^{R_{sd}} = 1$, otherwise $lb_{ij\lambda}^{R_{sd}} = 0$, $\forall e_{ij} \in E$, $\forall R_{sd} \in R$.

$lp_{ij\lambda}^{R_{sd}}$: if the wavelength λ along the link e_{ij} has already been allocated as the primary channel for R_{sd}, $lp_{ij\lambda}^{R_{sd}} = 1$, otherwise $lp_{ij\lambda}^{R_{sd}} = 0$, $\forall e_{ij} \in E$, $\forall R_{sd} \in R$.

$ld_{ij\lambda}$: if $lb_{ij\lambda}^{R_{sd}} = 1$ and $lp_{ij\lambda}^{R_{sd}} = 1$, $ld_{ij\lambda} = 1$, otherwise $ld_{ij\lambda} = 0$.

P_R_{sd}: primary light-path vector of the R_{sd}.

B_R_{sd}: backup light-path vector of the R_{sd}.

n_{ija}: number of the free wavelengths along the link e_{ij}, $\forall e_{ij} \in E$.

n_{ijc}: number of wavelengths supported by the link e_{ij}, here, $n_{ijc} = W$.

n_{ijb}: number of wavelengths used as the backup channels along the link e_{ij}, $\forall e_{ij} \in E$.

n_{ijp}: number of wavelengths used as the primary channels along the link e_{ij}, $\forall e_{ij} \in E$.

$dl_{R_{sd}}$: delay constraint of the R_{sd}, $\forall R_{sd} \in R$.

$jt_{R_{sd}}$: delay jitter constraint of the R_{sd}, $\forall R_{sd} \in R$.

$ls_{R_{sd}}$: failure rate constraint of the R_{sd}, $\forall R_{sd} \in R$.

The corresponding mathematical model of the proposed fault-tolerant QoS unicast routing scheme in this paper is described as follows:

Objective:

$$minimize(Op) \tag{1}$$

s.t.

$$\sum_{e_{ij} \in P\_R_{sd}} d(e_{ij}) + \sum_{v_i \in P\_R_{sd}} t(v_i) \leq dl_{R_{sd}} \tag{2}$$

$$\sum_{e_{ij} \in B\_R_{sd}} d(e_{ij}) + \sum_{v_i \in B\_R_{sd}} t(v_i) \leq dl_{R_{sd}} \tag{3}$$

$$\sum_{v_i \in P\_R_{sd}} dj(v_i) \leq jt_{R_{sd}} \tag{4}$$

$$\sum_{v_i \in B\_R_{sd}} dj(v_i) \leq jt_{R_{sd}} \tag{5}$$

$$1 - \prod_{e_{ij} \in P\_R_{sd}} (1 - l(e_{ij})) \leq ls_{R_{sd}} \tag{6}$$

$$1 - \prod_{e_{ij} \in B\_R_{sd}} (1 - l(e_{ij})) \leq ls_{R_{sd}} \tag{7}$$

here,

$$Op = \sum_{e_{ij} \in E} \sum_{\lambda \in W} \sum_{R_{sd} \in R} ld_{ij\lambda} \cdot lb_{ij\lambda}^{R_{sd}} \tag{8}$$

3 Design of the Proposed Scheme

In this paper, a hybrid intelligent algorithm taking advantage of both ant-colony algorithm and genetic algorithm is adopted; the crossover operation of the genetic algorithm is introduced into the ant-colony algorithm. Due to its capability of reassembling the existing solutions, the crossover operation can help the ant colony algorithm to alleviate being trapped into local optima, and thus the capability of searching the optimum solution globally of the ant colony algorithm is improved.

With the preventive fault-tolerance strategy adopted, the ants are classified into two types accordingly: ant for primary light-path and ant for backup light-path.

3.1 QoS Light-Path Establishment Request Description

For each QoS light-path establishment request R_{sd}, it is denoted as follows: $< v_s, v_d, dl_{R_{sd}}, jt_{R_{sd}}, ls_{R_{sd}} >$

3.2 Ant Description

For each R_{sd}, generate n pairs of the ant and n is a positive integer. Each ant pair is composed of two ants: one for primary light-path and the other for backup light-path, used to establish the primary and backup QoS-constrained light-path respectively.

An ant has certain memory capability, denoted as follows:$< at.ID, at.tp, at.dl, at.jt, at.ls, at.A_{nd}, at.A_{cl}, at.dt >$, with its elements corresponding to identifier of the ant, type of the ant, delay of its traversed path by the ant, delay jitter of its traversed path by the ant, failure rate of its traversed path by the ant, set of nodes of its traversed path by the ant, set of channels of its traversed path by the ant, destination node of the ant.

3.3 Expression and Update of the Pheromone

There are two types of pheromone in this paper. One type of pheromone is excreted by the primary light-path ant and the other by the backup light-path ant, which are denoted as ψ^p and ψ^b respectively.

Suppose that the range of pheromone thickness for ψ^p and ψ^b is $[\psi_{min}, \psi_{max}]$, and the initial value of pheromone thickness on each channel is set to be ψ_{max}. When the value of the pheromone thickness is out of the above range, it is set to be ψ_{min}or ψ_{max}. Use $\psi^p_{e^\lambda_{ij}}(t)$ and $\psi^b_{e^\lambda_{ij}}(t)$ to denote the pheromone thickness of ψ^p and ψ^b on the channel λ of the link e_{ij} at time t respectively, and use $\Delta\psi^p_{e^\lambda_{ij}}(t, t+1)$ and $\Delta\psi^b_{e^\lambda_{ij}}(t, t+1)$ to denote the variation of the pheromone thickness of ψ^p and ψ^b on the channel λ of the link e_{ij} during $[t, t+1]$.

The update of pheromone is divided into two types: local and global. When the primary light-path ant k and the backup light-path ant $k^{'}$ walk through the channel λ of the link e_{ij}, the pheromone thickness is updated locally, and the update rule is defined as follows:

$$\psi^p_{e^\lambda_{ij}}(t+1) = (1-\rho)\psi^p_{e^\lambda_{ij}}(t) + \phi \cdot \Delta\psi^p_{e^\lambda_{ij}}(t, t+1) \tag{9}$$

$$\psi^b_{e^\lambda_{ij}}(t+1) = (1-\rho)\psi^b_{e^\lambda_{ij}}(t) + \phi \cdot \Delta\psi^b_{e^\lambda_{ij}}(t, t+1) \tag{10}$$

$$\Delta\psi^p_{e^\lambda_{ij}}(t, t+1) = \begin{cases} n_{ija} + \sigma \cdot n_{ijb} & s(v_i) = 1 \\ \sigma \cdot (1 - lb^{R_{sd}}_{ij\lambda}) & s(v_i) = 0 \end{cases} \tag{11}$$

$$\Delta\psi^b_{e^\lambda_{ij}}(t, t+1) = \begin{cases} n_{ija} + \sigma \cdot (n_{ijb} + n_{ijp}) & s(v_i) = 1 \\ \sigma \cdot (2 - lb^{R_{sd}}_{ij\lambda} - lp^{R_{sd}}_{ij\lambda}) & s(v_i) = 0 \end{cases} \tag{12}$$

where k, $k^{'} = 1, 2, ..., n$, ρ is the volatile factor of pheromone, $0 < \rho < 1$; ϕ is the update coefficient; σ is the sharing factor, $0 < \sigma < 1$. Obviously, the variation of the pheromone thickness of λ relies on $\Delta\psi^p_{e^\lambda_{ij}}(t, t+1)$ when the primary light-path ant k walks through the channel λ on the link e_{ij}. The value of $\Delta\psi^p_{e^\lambda_{ij}}(t, t+1)$ relies on whether the node v_i has the wavelength conversion capability and the usage of the wavelengths of the link e_{ij} starting from v_i. If its thickness value is less than the specified threshold, the pheromone on the channel λ is on the state of volatilization; otherwise, it is on the state of enhancement. When the backup light-path ant $k^{'}$ walks through the channel λ on the link e_{ij}, its situation is similar to that of the primary light-path ant k.

When the primary light-path ant k and the back up light-path ant $k^{'}$ arrive at the destination node after n times (assume the primary light-path ant k arrives at the destination node after n_1 times and the back up light-path ant $k^{'}$ arrives at the destination node after n_2 times, then $n = max\{n_1, n_2\}$), whether the chosen primary and backup light-path pair is good or bad should be judged (see section 3.6): if it is good, the pheromone is updated globally; otherwise, do nothing. The global update rule is defined as follows:

$$\psi^{p}_{e^{\lambda}_{ij}}(t+n) = (1-\rho)\cdot\psi^{p}_{e^{\lambda}_{ij}}(t) + \phi\cdot\Delta\psi_{e^{\lambda}_{ij}}(t, t+n) \tag{13}$$

$$\psi^{b}_{e^{\lambda}_{ij}}(t+n) = (1-\rho)\cdot\psi^{b}_{e^{\lambda}_{ij}}(t) + \phi\cdot\Delta\psi_{e^{\lambda}_{ij}}(t, t+n) \tag{14}$$

$$\Delta\psi_{e^{\lambda}_{ij}}(t, t+n) = \begin{cases} \frac{Q}{Op_k+1} & e^{\lambda}_{ij} \text{ walked through by the } k\text{th ant pair} \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

where Q is a constant and Op_k denotes the number of orphan connections generated by the kth ant pair.

3.4 Transfer Probability of the Ant

When one ant needs to transfer to the next hop channel, it is attracted by the pheromone left on the channel by those congenerous ants at the same time it is excluded by the pheromone left on the channel by the egregious ants and the congenerous ants.

Use $\alpha^{pk}_{e^{\lambda}_{ij}}(t)$ to denote the attraction strength to the primary light-path ant k by the channel e^{λ}_{ij} at time t, $\alpha^{bk^{'}}_{e^{\lambda}_{ij}}(t)$ the attraction strength to the backup light-path ant $k^{'}$, $\beta_{e^{\lambda}_{ij}}(t)$ the exclusion strength to the primary light-path ant k and also to the backup light-path ant $k^{'}$ respectively. Their definitions are as follows:

$$\alpha^{pk}_{e^{\lambda}_{ij}}(t) = \psi^{p}_{e^{\lambda}_{ij}}(t) \tag{16}$$

$$\alpha^{bk^{'}}_{e^{\lambda}_{ij}}(t) = \psi^{b}_{e^{\lambda}_{ij}}(t) \tag{17}$$

$$\beta_{e^{\lambda}_{ij}}(t) = \psi^{p}_{e^{\lambda}_{ij}}(t) + \psi^{b}_{e^{\lambda}_{ij}}(t) \tag{18}$$

According to formula (16-18), the attraction strength to one ant by one channel relies on the thickness of the pheromone left by its congenerous ants, however, the exclusion strength to one ant by one channel relies on the thickness of the pheromone left by its egregious ants and its congenerous ants simultaneously. In fact, the exclusion to one ant by one channel mainly comes from its egregious ants when the pheromone thickness of the congenerous ants is weak, while the exclusion from its congenerous ants gets enhanced gradually with the enlargement of their corresponding pheromone thickness (this means that the channel becomes crowded gradually with its load increased).

The probability of the primary light-path ant k and the backup light-path ant $k^{'}$ selecting the link e_{ij} as the next hop at time t, called as its transfer probability, is defined as follows:

$$P_{e_{ij}^{\lambda}}^{pk}(t) = \begin{cases} 0 & lp_{ij\lambda}^{R_{sd}} = 0 \\ \frac{\alpha_{e_{ij}^{\lambda}}^{pk}(t)/[\beta_{e_{ij}^{\lambda}}(t)]^{\varepsilon}}{\Sigma_{e_{ij}^{\lambda} \in A}[\alpha_{e_{ij}^{\lambda}}^{pk}(t)/[\beta_{e_{ij}^{\lambda}}(t)]^{\varepsilon}]} & \text{otherwise} \end{cases} \tag{19}$$

$$P_{e_{ij}^{\lambda}}^{bk^{'}}(t) = \frac{\alpha_{e_{ij}^{\lambda}}^{bk^{'}}(t)/[\beta_{e_{ij}^{\lambda}}(t)]^{\varepsilon}}{\Sigma_{e_{ij}^{\lambda} \in A^{'}}[\alpha_{e_{ij}^{\lambda}}^{bk^{'}}(t)/[\beta_{e_{ij}^{\lambda}}(t)]^{\varepsilon}]} \tag{20}$$

where A and $A^{'}$ are the set of channels from which the primary light-path ant k and the backup light-path ant $k^{'}$ can select as the next hop respectively, and ε is a constant.

3.5 Expression of the Solution

Use the array to denote the solution. Number the links in the graph $G(V, E)$. The ith element of the array corresponds to the ith link; if the light-path passes through a wavelength channel on the link, the corresponding element value to the link is set to be the number of the wavelength used on the link; otherwise, its value is set to be -1.

3.6 Judgment of the Generated Primary and Backup Light-Path Pair

Whether one generated primary and backup light-path pair is good or bad should be judged after one ant pair has finished the routing procedure. There are three conditions to be considered: for the same light-path establishment request, its primary and backup light-path must be link-disjoint; for different light-path establishment requests, their primary light-paths can not share the same channel; for different light-path establishment requests, their backup light-path can share the same channel only when their primary light-paths are link-disjoint.

If one primary and backup light-path pair generated for one light-path establishment request satisfies the above three conditions simultaneously, it is considered good.

3.7 Fitness Function

Select the best one from all good primary and backup light-path pairs as the up-to-now best solution after the ant colony completed one routing cycle. The selection criterion is that the current number of the orphan connections in the network is smallest. Thus, the fitness function is defined as follows:

$$f(V_s, V_d) = \sum_{e_{ij} \in E} \sum_{\lambda \in W} \sum_{R_{sd} \in R} ld_{ij\lambda} \cdot lb_{ij\lambda}^{R_{sd}} \tag{21}$$

Obviously, the smaller the value of the fitness function, the better the primary and backup light-path pair.

3.8 Crossover Operation

After the ant colony completed one cycle, select two pairs from all good primary and backup light-path pairs randomly to do single-point crossover operation. The specific crossover point is chosen by the specified crossover probability. If the newly generated primary and backup light-path pair after crossover is better than the up-to-now best solution, update it.

3.9 Procedure of the Proposed Algorithm

The procedure of the proposed hybrid intelligent preventive fault-tolerant QoS unicast routing algorithm is described as follows:

STEP1: Set the parameter values for each node and link in graph $G(V, E)$, including range of the pheromone thickness and its initial value. Set the maximum cycling times to be $NCMax$, the initial value of the cycling counter NC to be 0, and the initial fitness value F_V of the up-to-now best solution to be $+\infty$.

STEP2: Generate n pairs of the ants for the QoS light-path establishment request R_{sd}. For each ant, set $at.dl = 0$, $at.jt = 0$, $at.ls = 0$, $at.A_{nd} = \{v_s\}$, $at.A_{cl} = \Phi$, $at.dt = v_d$, assign $at.ID$, set $at.tp$, and put it at v_s; set the timer $t = 0$.

STEP3: if $NC < NCMax$, go to STEP4; otherwise, output the up-to-now best solution, the algorithm ends.

STEP4: Select one ant pair, the primary and backup light-path ant choose the channel e_{ij}^{λ} with the maximum transfer probability as the next hop according to the formula (19) and (20) respectively (if there are more than one such channels, choose one from them randomly), then transferring from v_i to v_j. Let $t = t + 1$, update each domain value of the ant as follows:

$at.dl = at.dl + d(e_{ij}) + t(v_i)$;
$at.jt = at.jt + dj(v_i)$;
$at.ls = 1 - (1 - at.ls) \cdot (1 - l(e_{ij}))$;
$at.A_{nd} = at.A_{nd} \bigcup \{v_j\}$;
$at.A_{cl} = at.A_{cl} \bigcup \{e_{ij}^{\lambda}\}$;

If $at.dl > dl_{R_{sd}}$ or $at.jt > jt_{R_{sd}}$ or $at.ls > ls_{R_{sd}}$, kill the ant pair; otherwise, update the pheromone thickness locally according to the formula (9) and (10).

STEP5: If all n ant pairs arrived at v_d or were killed, go to STEP6; otherwise, go to STEP4.

STEP6: Judge whether the generated primary and backup light-path pairs are good or bad according to section 3.6. If no good primary and backup light-path pair exists, go to STEP8; otherwise, update the pheromone thickness globally according to the formula (13) and (14) for all good primary and backup light-path pairs, calculate their fitness values, if their minimum is smaller than F_V, update

F_V and the up-to-now best solution with this minimum and the corresponding primary and backup light-path pair.

STEP7: According to section 3.8, select two pairs randomly from all good primary and backup light-path pairs to do crossover operation. If the newly generated pair is good and it is better than the up-to-now best solution, update the latter and its F_V with the former and its fitness value.

STEP8: Kill all ants, let $NC = NC + 1$, then go to STEP3.

4 Simulation

Simulated implementation of the above-proposed scheme has been done based on NS2 platform [12] and its performance evaluation has been done over some actual and virtual network topologies (such as CERNET, CERNET2 and NSFNET) [8].

Comparison on the number of the generated orphan connections in the network between the fault-tolerant routing scheme based on the classical ant colony algorithm [13] and the proposed scheme in this paper is shown in Fig.1. The improved performance of the proposed scheme comes from the introduction of the crossover operation of genetic algorithm into ant colony algorithm.

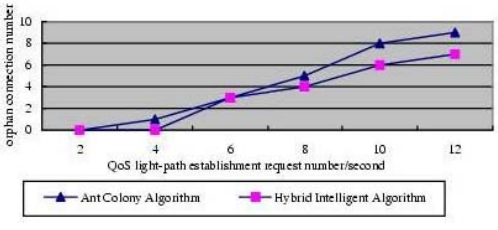


Fig. 1. Comparison on the orphan connection number

Comparison on the blocked rate of QoS light-path establishment requests among the fault-tolerant routing schemes based on backup multiplexing and no multiplexing respectively [13] and the proposed scheme based on primary-backup multiplexing is shown in Fig.2. The proposed scheme has the lowest blocked rate due to its highest resource utilization.

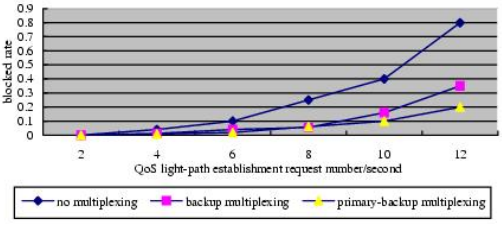


Fig. 2. Comparison on the blocked rate

5 Conclusion

IP over DWDM optical Internet is one of the main networking techniques for NGI backbone. In this paper, with preventive fault-tolerance strategy and primary-backup multiplexing mechanism adopted, a fault-tolerant QoS unicast routing scheme based on a hybrid intelligent algorithm is proposed by introducing crossover operation of genetic algorithm into ant colony algorithm. The proposed scheme establishes the primary and backup QoS-constrained light-path pair for the QoS light-path establishment request with the objective of minimizing the number of the generated orphan connections in the IP over DWDM optical Internet. Simulation results have shown that the proposed scheme is both feasible and effective with better performance.

References

1. Sridharan, M., Salapaka, M. V., Somani, A. K.: A Practical Approach to Operating Survivable WDM Networks. IEEE Journal on Selected Areas in Communications, Vol.20, No.1. (2002) 34–46
2. Wang, X. W., Li, J., Lin, W. H., Huang, M.: Research on Fault-Tolerant Routing Mechanism in IP/DWDM Optical Internet. Journals of Northeastern university, Vol.25, No.11. (2004) 1054–1057
3. Modiano, E., Narula-Tam, A.: Survivable Light-Path Routing: a New Approach to the Design of WDM-Based Networks. IEEE Journal on Selected Areas in Communications, Vol.20, No.4. (2002) 800–809
4. Anandkrishna, P., Miguel, A. L., Ibrahim, H., et al: Improving Bandwidth Efficiency in Fault-Tolerant IP over Optical Mesh Networks. International Journal of Network Management, Vol.14, No.1. (2004) 19–27
5. Wang, X. W., Cai, S. X., Huang, M.: A Reactive Fault-Tolerant Integrated QoS Unicast Routing Scheme in IP over DWDM Optical Internet. Journals of Northeastern University, Accepted
6. Wang, Z., Crowcroft, J.: Quality of Service Routing for Supporting Multimedia Applications. IEEE Journal on Selected Areas in Communications, Vol.14, No.7. (1996) 1288–1294
7. Mohan, G., Somani, A. K.: Routing Dependable Connections with Specified Failure Restoration Guarantees in WDM Networks. In Proceedings of IEEE Infocom, Vol.3. (2000) 1767–1770
8. Dorigo, M., Caro, G., Gambardella, L. M.: Ant Algorithms for Discrete Optimization. Massachusetts Institute of Technology Artificial Life, Vol.5. (1999) 137–172
9. Graham, R., Levine, J.: A Hybrid Ant Algorithm for Scheduling Independent Jobs in Heterogeneous Computing Environments. American Association for Artificial Intelligence (www.aaai.org), (2004)
10. Shi, R. F., Zhou, H., Tan X. W.: Progressive Multi-Objective Genetic Algorithm. Theory and Practice of System Engineering, Vol.12 (2005) 48–57
11. George, N.:Routing and Wavelength Assignment in Optical WDM Networks, Wiley and Sons, (2001) 34–82
12. Wei, J. L., Xiao, Y. H., Zhang, C.:Architecture of NS and Its Extending, Computer Simulation, Vol.21, No.8. (2004) 179–182
13. Yu, Y.: Research and Simulated Implementation on Ant-Colony-Algorithm-Based Fault-Tolerant Routing Mechanisms in IP over DWDM Optical Internet. Northeastern University, Shengyang, (2005)

From XML Specifications to Parallel Programs*

Ignacio Peláez, Francisco Almeida, and Daniel González

Departamento de Estadística, I. O. y Computación
Universidad de La Laguna, c/ Astrofísico F. Sánchez s/n
38271 La Laguna, Spain
ignacio.pelaez@gmail.com, falmeida@ull.es, dgonmor@ull.es

Abstract. Skeleton-based libraries are considered as one of the alternatives for reducing the distance between end users and parallel architectures. We propose a general development methodology that allows for the automatic derivation of parallel programs assuming the existence of general structures as the skeletons. We propose the introduction of a new, high level abstraction layer that allows the user to extract problem specifications from particular skeleton languages or libraries. The result is a tool that allows for the generation of parallel codes from successive transformations to this high level specification without any loss of efficiency. We apply the technique to the automatic generation of parallel programs for Dynamic Programming Problems.

1 Introduction

A widespread research methodology among scientists is based on developing a mathematical formulation that conceptually provides a solution to a problem. However, it is also common for the problem to remain partially unsolved if the scientist is not able to transform this conceptual solution into a computer program. Researchers usually bridge this gap by using software tools like mathematical solvers, or even by writing their own code. This gap is even more evident in the case of parallel computers, where much more effort is required by the user.

A substantial effort has been made in recent decades to reduce the distance between parallel architectures and end users. As stated in [2], skeletal programming has emerged as an alternative and has helped simplify programming, enhance portability and improve performance. Such systems conceal the parallelism from the programmer and are characterized by being embedded entirely into a functional programming language, or for integrating imperative code within a skeletal framework in a language or library. Some of these approaches can be seen in [4], [8], [3], [1], [6], [11], [5], [2]. The underlying idea of separating the specification of a problem, or an algorithm, from implementation details that are hidden to the user is present in all the proposals.

As also stated in [2], skeletal programming has yet to make a substantial impact on mainstream methods in parallel applications programming. It is safe to say that in many cases, the end user of many algorithmic skeletons, like those

* Supported by MCyT projects TIN2005-09037-C02-01.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 267–277, 2006.
© Springer-Verlag Berlin Heidelberg 2006

derived from algorithmic techniques (Divide and Conquer, Simulated Annealing, etc.), is not a programmer but a scientist, say a biologist or a physicist, or a mathematician. The potential use of this kind of tool is strongly conditioned by its ease of use. Those who lack expertise in object or functional oriented programming, or who have no programming knowledge at all, should be able to apply them.

Based on XML specifications, we propose to introduce a new abstraction level that contributes to a higher separation between the specification of a problem and the implementation details. Skeletons act as an intermediate level between the specification made by the user and the parallel implementation, so that the user describes the problem through a general model and successive transformations convert the model into the instance code for a skeleton. The distance between the scientific knowledge and the specification is bridged by the scientist (not a programmer), while the distance between the specification and the executable code is bridged by the machine. Since the code generated is based on skeleton tools, the approach still maintains the advantages of the skeletal development methodology while providing significant benefits for the scientist:

- No need for codification. The user specifies the problem and does not codify the algorithm.
- Independence from specific programming languages or skeleton libraries. Once the problem has been specified, it can be transformed into several implementation proposals.
- Delivery of new applications due to the rapid development time.
- Improved application quality.
- Increased use of parallel architectures by non-expert users.
- Rapid inclusion of emerging technology into their systems. New transformers can be delivered when needed.

This paper makes two primary contributions: a general development methodology for deriving parallel programs automatically, starting from a very high level problem specification, and an application example in generating parallel programs for Dynamic Programming (DP from now on) problems.

Although we focus on a specific scientific domain, the parallelization of DP Problems, from a general point of view the technique presented here can be considered as a particular instance of the MDA (Model Driven Architecture) [9] general software engineering development methodology, and can be applied to many other scientific domains. We will show how this methodology can be applied, without any loss of efficiency, to parallel architectures. We also require the project to adhere to general standards as much as possible; for that reason, we follow the W3C [20] recommendations along every transformation step.

This paper is structured as follows: in Section 2 we introduce the software architecture of the proposed methodology and show how the technique is applied for the particular case of Dynamic Programming problems, and in Section 3 an XML specification for Dynamic Programming is presented. The paper finalizes with some concluding remarks and future lines of work.

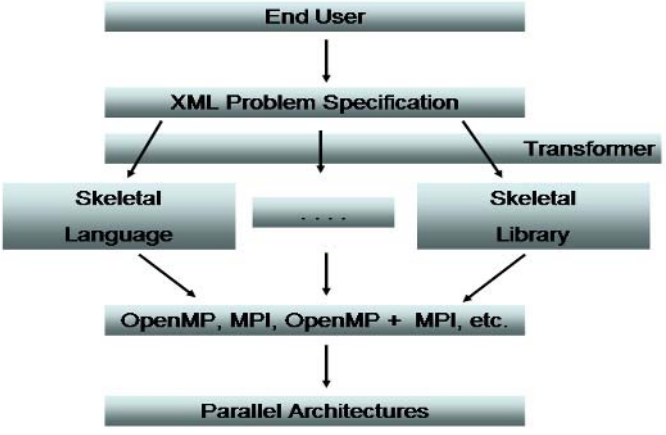


Fig. 1. Software Architecture for the Methodological Approach

2 Software Architecture

We propose the introduction of a new abstraction layer between the user and the skeletons so as to separate the fundamental logic behind a problem specification from the specifics of the particular middleware that implements it (the skeleton instance). The advantages of this approach were listed in Section 1. This new layer should be closer to the scientific notation and should also be simple enough to generate the appropriate code. In general, the use of XML as the specification language is not mandatory; however, technologies based on XML specifications have proven to be interesting standard alternatives for transformation into different formats. Although we use this specification to generate C++ code, it could be used to generate WSDL [19] and web services applications.

The software architecture of our transformation methodology is presented in Figure 1. The layer between the end user and the skeletal libraries and languages tries to span the significant gap for those scientists who are not directly involved with programming. In order to provide access to data coming from an XML document, an analysis of the document and its decomposition into nodes and recognizable pieces through a standardized API is needed. The proper linking of this analysis with the later processing can be achieved by following the DOM (Document Object Model) [15] recommendation of the W3C. DOM is a set of specifications for developing standards that allow programming languages to interact with documents. DOM manipulates the whole document as a tree and XML data are provided as nodes.

The elements needed to apply the methodology can be summarized as:

- A skeleton that can be architecture dependent.
- A specification language describing the domain of the application, independent of the architecture.
- A transformer of data documents expressed in that language as instances for the skeleton.

The transformation from XML documents into parallel programs could be done directly without using the skeleton layer; however, we consider the skeleton layer necessary as a way of separating the parallel implementation, from the specification of a solution to a problem using a sequential programming language. The skeleton introduces several advantages, such as modularity, ease of use, portability, etc... The sequential skeleton can be ported to any parallel machine and the appropriate parallel code can be generated if necessary.

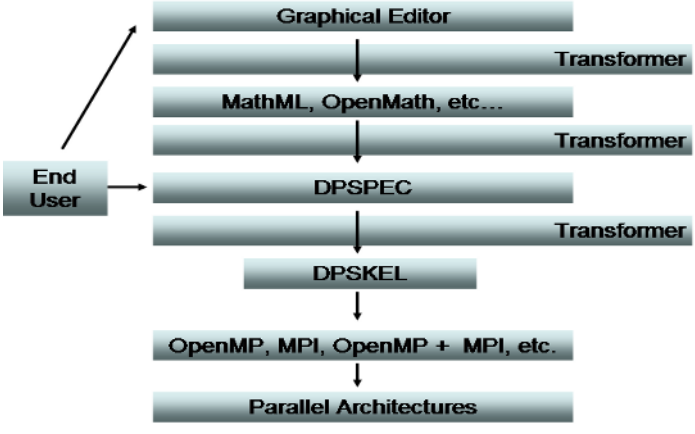


Fig. 2. Software Architecture for the Dynamic Programming Implementation

Figure 2 shows how we have implemented this methodology for the particular case of Dynamic Programming problems. The automatic parallelization of DP problems is achieved by making explicit transformations on the DPSPEC data file (our specification language, described in Section 3) containing the XML description of a Dynamic Programming equation. The XML description of the formula is converted into a specific instantiation of the DPSKEL C++ skeleton (actual release of the Dynamic Programming Mallba skeleton [10]) to solve the problem in question. The C++ code generated for the Dynamic Programming skeleton is the same as would be generated by an experienced C++ programmer, so no loss of efficiency is introduced during the process. This transformation step involves an analysis of the Dynamic Programming functional equation. We have developed a DPSPEC to DPSKEL transformer. Our transformer performs a DOM parsing of the XML functional equation to produce the proper DPSKEL C++ required classes. We have not developed a general parser for mathematical equations; instead, the DOM tree is parsed so that the nodes are matched with expected values on Dynamic Programming recurrences (operators, variables, constants, etc,...). Once a value is found, the proper C++ code is generated for it. The parser has been developed using the Xerces-C++ library [7]. Figure 3 shows a section of the parser that generates the C++ code of the Evalua DPSKEL method for a `<cond>` DPSPEC element. Typically, a Dynamic Programming equation is expressed as a piecewise equation, where a condition must be tested before the

function is evaluated. We use the label `<cond>` to express such a condition. Note in Figure 3 the use of Xerces-C++ library functions, such as `getFirstChild()` and `getNextSibling()`, to traverse the DOM tree. We have developed the `X2C_Exp.Exp()` method to evaluate expressions recursively.

```
.... omitted code
// Generate header for the method Evalua of class State
Buffer.write("void State::Evalua(int stage, int index)\n");
Buffer.write("{\n");
Buffer.write("\tint v0, v1;\n");
Buffer.write("\tState st0(pbm, sol, table);\n\n");

 // Parse a condition
 //  <cond>
 //     #Exp0
 //     #Exp1
 //  </cond>
 //  The former element is translated as
 //  if (#Exp0) {
 //      v0 = #Exp1;
 //  }
 if (!strcmp(name, "cond")) {
     do {
        strcpy(name, (const char *)XMLString::transcode(p->getNodeName()));
        if (!strcmp(name, "cond")) {
            Buffer.write("\tif ");          // Generate if code
            X2C_Exp.Exp(p->getFirstChild());
            Buffer.write(" {\n");
            Buffer.write("\t\tv0 = ");
            X2C_Exp.Exp(p->getFirstChild()->getNextSibling());
            Buffer.write(";\n\t}\n");
         } else
            break;
         p = p->getNextSibling();
      } while(p != 0);
 } else
     X2C_Exp.Exp(p);
 // Generate the assignment of the evaluated state and the subsequent insertion in table
 Buffer.write("\tst0.setvalue(v0);\n");
 Buffer.write("\ttable.PUT_STATE(st0, stage, index);\n");
 Buffer.write("}\n\n");
 // Write the generated code to the required DPSKEL C++ data file
 if ((fevalua = fopen("dp.req.cc","w")) == NULL)
     return (-1);
 fprintf(fevalua, "%s", Buffer.buf());
 fclose(fevalua);
 return (0);
```

Fig. 3. DOM parser for an DPSPEC XML data file (element `<problem>`)

The Evalua (void State::Evalua(int stage, int index)) method, required for users of DPSKEL, includes a C++ specification of the recurrence equation. The code in Figure 4 shows the method for the optimal matrix parenthesization (Table 1 and Equation 1). Note that the DOM parser shown in Figure 3 generates C++ code since we are interested in the specific DPSKEL C++ implementation. A different parser can be developed that generates code for any other programming language if needed.

```
void State::Evalua(int stage, int index)
{
        int v0, v1;
        State st0(pbm, sol, table);

        if (stage == index) {
                v0 = 0;
        }
        if (stage != index) {
                v0 = INT_MAX;
        for( int k = stage; k <= (index - 1); k++) {
                v1 = (table.GET_STATE(stage, k).get_value() +
                     table.GET_STATE((k + 1), index).get_value() +
                     (r[(stage - 1)] * r[k] * r[index]));
                if (v1 < v0)
                        v0 = v1;
        };
        }
        st0.setvalue(v0);
        table.PUT_STATE(st0, stage, index);
}
```

Fig. 4. Evalua method generated for the Optimal Matrix Parenthesization (see Table 1 and Equation 1)

However, scientists typically represent the functional equations as mathematical expressions, like those shown in Table 1, by using their favorite equation editor (Latex, OpenOffice, etc.). Therefore, a new transformation is implicitly involved in the process: the conversion of the mathematical equation to the XML specification. Currently, the transformation between a mathematical formula and an XML specification is provided automatically for many popular MathML (the W3C recommendation for describing mathematics as a basis for machine to machine communication) [17], [16], software editors, for example, EdiTex, OpenOffice, Scientific Word, sMArTH, etc. (see [18] for a detailed MathML software list). All of them generate a MathML document for a given equation. DPSPEC is compatible with MathML and the conversion between a MathML document into a DPSPEC document can be achieved through a preprocessing step.

3 DPSPEC: An XML Specification for Dynamic Programming Problems

In this section we show the XML specification that we have developed for DP problems. Several XML specifications have been proposed to describe the semantics of

Table 1. Examples of Dynamic Programming recurrences

Problem	Recurrence	DP Category
0/1 Knapsack	$f_{kc} = max\{f_{k-1c}, f_{k-1c-w_k} + p_k\}$	Serial Monadic
Longest Common Subsequence	$f_{ij} = f_{i-1j-1} + 1 \;\; if \;\; x_i = y_j; \;\; or$ $f_{ij} = max\{f_{ij-1}, f_{i-1j}\} \;\; if \;\; x_i \neq y_j$	Nonserial Monadic
Floyd's All-Pairs Shortest-Paths	$d_{ij}^k = min\{d_{ij}^{k-1}, d_{ik}^{k-1} + d_{kj}^{k-1}\}$	Serial Polyadic
Optimal Matrix Parenthesization	$c_{ij} = min\{c_{ik} + c_{k+1j} + r_{i-1} r_k r_j\}$	Nonserial Polyadic

mathematical expressions: MathML [17], OpenMath [13], OMDoc [12], XDF [14]. Any of them could be used to specify the DP functional equation; however, we decided to develop our own specification adapted to DP problems. The reason behind our decision was two-fold: to reduce the elements to a minimum, and to introduce some changes in the structure for specific elements appearing in classical specifications. These design issues make the subsequent parsing easier and allow for a better semantic analysis to detect data dependencies, all while adhering as closely as possible to the user defined equation. The semantic analysis determines the traversing mode of the DP table. DPSPEC brings together the elements to describe piecewise defined functions, simple variables and vectors, arithmetic, logical, relational and max-min operators, and iterators. Table 2 summarizes the elements available in the DPSPEC language. The proper syntaxis and semantics of DPSPEC have been defined as an XSD schema (omitted in the paper due to space constraints).

Table 2. Current elements available in DPSPEC

Element	Operation
Main element	`<problem>`
Logical functions	`<and>, <or>, <not>`
Conditional	`<cond>`
Relational operators	`<lt>, <le>, <gt>, <ge>, <eq>, <ne>`
Binary mathematical operators	`<minus>, <divide>, <power>`
N-ary mathematical operators	`<plus>, <times>, <max>, <min>`
N-ary iterative operators	`<imax>, <imin>`
States	`<state>`
Variables	`<ci>`
Constants	`<cn>`
User defined functions	`<functiondef>, <function>`
User defined vectors	`<vectordef>, <vector>`

DPSPEC has been used to specify DP problems appearing in Table 1. As an example to illustrate the use of DPSPEC, we will consider the functional equation for the multiplication parenthesization problem (Equation 1). This is

a paradigmatic problem and its recurrence equation represents a wide range of Dynamic Programming applications. Note that the equation appears represented as a piecewise function. The variables i and j will be mapped as the variables *stage* and *index* of the method Evalua of the class State in DPSKEL. Figure 6 shows the XML specification using DPSPEC for Equation 1. The `<cond>` element allows us to describe each one of the conditions of the equation. Thanks to the availability of assistant XML editors, non-expert programmers can generate DPSPEC XML specifications more readily than sequential C programs can. Moreover, as stated in Section 2, the DPSPEC code can also be directly generated from equation editors.

$$c(i,j) = \begin{cases} 0 & \text{if j = i, } 0 < \text{i} \leq \text{n} \\ min_{i \leq k < j}\{c(i,k) + c(k+1,j) + r_{i-1} \cdot r_k \cdot r_j\} & \text{if } 1 \leq i < j \leq n \end{cases} \quad (1)$$

The generation of the parallel code is quite straightforward and could be detailed in these steps:

The user defines the equation in a math editor and executes a transformation (several transformations from standard Mathematical specifications into DPSPEC will be provided). In this step the user has his problem as a DPSPEC document generated by the transformation.

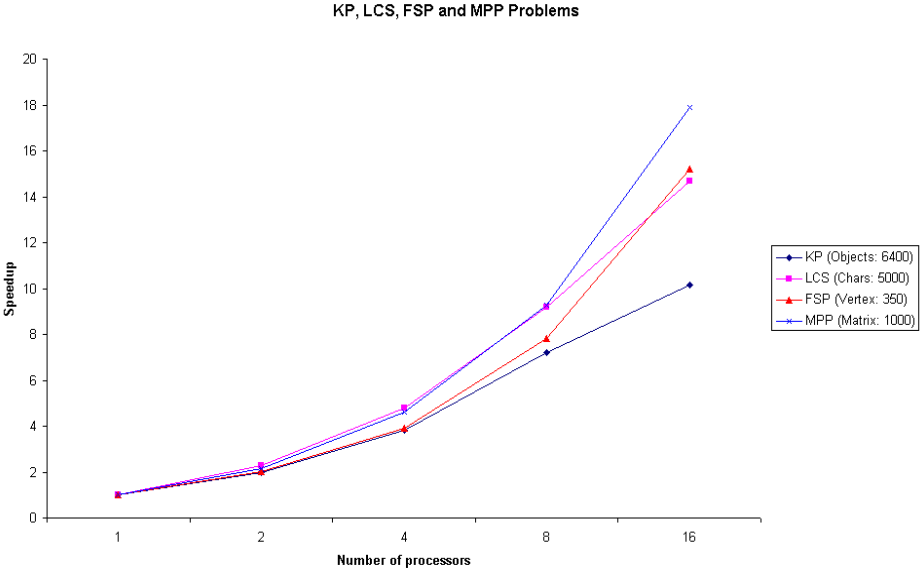


Fig. 5. Speedups obtained for Knapsack Problem (KP), Longest Common Subsequence (LCS), Floyd All-Pairs Shortest-Paths algorithm (FSP) and Optimal Matrix Parenthesization Problem (MPP)

```
<?xml version = '1.0' encoding = 'utf-8'?>
<problem>
  ....

  <!-- Declare a vector -->
  <vectordef load="xmlfile" type="int" name="r" vars="1" />
  <!-- c(stage, index) = {
                          0                                 if (stage = index),
              min{c(stage, k) + c(k + 1, index) + r(stage-1) * r(k) * r(index)}
                                                            if (stage <> index)
                       } -->

  <cond> <!-- 0 if (stage = index) -->
    <eq>
      <ci>stage</ci>
      <ci>index</ci>
    </eq>
    <cn>0</cn>
  </cond>

  <!-- Evalua c(stage, index) -->
  <cond> <!-- min{c(stage, k) + c(k + 1, index) + r(stage-1) * r(k) * r(index)}  if (stage <> index)} -->

    <ne>                    <-- if (stage <> index)} -->
      <ci>stage</ci>
      <ci>index</ci>
    </ne>
    <imin>                  <-- iterative min from k = stage to index - 1 -->
      <var>k</var>
      <startinterval>
        <ci>stage</ci>
      </startinterval>
      <endinterval>
        <minus>
          <ci>index</ci>
          <cn>1</cn>
        </minus>
      </endinterval>
      <plus>                <-- c(stage, k) + c(k + 1, index) + ... -->
        <state>
          <ci>stage</ci>
          <ci>k</ci>
        </state>
        <state>
          <plus>
            <ci>k</ci>
            <cn>1</cn>
          </plus>
          <ci>index</ci>
        </state>
        <times>             <-- ... + r(stage - 1) * r(k) * r(index) -->
          <vector name="r" >
            <minus>
              <ci>stage</ci>
              <cn>1</cn>
            </minus>
          </vector>
          <vector name="r" >
            <ci>k</ci>
          </vector>
          <vector name="r" >
            <ci>index</ci>
          </vector>
        </times>
      </plus>
    </imin>
  </cond>
</problem>
```

Fig. 6. XML description of the Multiplication Parenthesization problem using DPSPEC

In the next step the user will execute the parser (Figure 3) to transform the DPSPEC document into code for the library used (in this case for library DPSKEL in C++). In this step the user has the needed code for the skeleton used.

Note that until now the steps are independent from the final architecture, so this steps could be provided in a general way for any problem, for example using web services in a remote computer.

Finally in the last step the user compiles the code generated for the library using the needed compiler and libraries for the target architecture. In this step the user has a parallel program for an specific architecture that solves his problem.

Since the time invested by the parser to generate the DPSKEL code is negligible, the time invested in generating a parallel code from the XML specification is comparable to the time invested in compiling any other parallel program. No extra overhead is introduced.

In terms of the efficiency of the generated code, Figure 5 shows the performance obtained with the DPSKEL-generated parallel code. Series of instances for the problems in Table 1 were randomly generated for each problem. The instances of the skeleton generated use the OpenMP library [10] on an RS-6000 SP IBM platform. The tool's performance was satisfactory in all the cases. We observed increasing speedups in all the cases when the number of processors was increased. Superlinear speedup was observed in some cases, likely due to improved cache use on the parallel versions.

4 Conclusions and Future Work

In summary, we proposed a general methodology that allows for the automatic generation of parallel programs. The methodology is based on the existence of general parallel programs, like skeletons, that can be generated from higher level specification languages such as, for example, XML-based specifications. The code generated is efficient since no overhead is introduced during the transformation steps. The technique was validated by applying it to the generation of parallel Dynamic Programming algorithms. Once an XML specification has been stated, transformations from/to many other languages can be developed at a reasonable cost. We are also interested in the development of new transformation tools from other languages to DPSPEC, as is the case with OpenMath, for example, and from DPSPEC to other languages such as WSDL to provide the interface for a web service application.

References

1. M. Aldinucci, S. Gorlatch, C. Lengauer, and S. Pelagatti. Towards parallel programming by transformation: The FAN skeleton framework. *Parallel Algorithms and Applications*, 16(2–3):87–122, 2001.
2. Murray Cole. Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. *Parallel Comput.*, 30(3):389–406, 2004.

3. Marco Danelutto and Massimiliano Stigliani. Skelib: Parallel programming with skeletons in c. In *Euro-Par '00: Proceedings from the 6th International Euro-Par Conference on Parallel Processing*, pages 1175–1184, London, UK, 2000. Springer-Verlag.
4. John Darlington, A. J. Field, Peter G. Harrison, Paul H. J. Kelly, D. W. N. Sharp, and Q. Wu. Parallel programming using skeleton functions. In *PARLE '93: Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe*, pages 146–160, London, UK, 1993. Springer-Verlag.
5. A. J. Dorta, J. A. González, C. Rodríguez, and F. de Sande. llc: A parallel skeletal language. *Parallel Processing Letters*, 13(3):437–448, 2003.
6. E. Alba et al. MALLBA: A library of skeletons for combinatorial optimisation (research note). In *Proceedings of the 8th International Euro-Par Conference*, volume 2400 of *LNCS*, pages 927–932, 2002.
7. The Apache Software Foundation. Xerces-c++. `http://xml.apache.org/xerces-c/`.
8. Daniel González-Morales, Francisco Almeida, F. Garcia, J. Gonzalez, Jose; L. Roda, and Casiano Rodríguez. A skeleton for parallel dynamic programming. In *Euro-Par '99: Proceedings of the 5th International Euro-Par Conference on Parallel Processing*, pages 877–887, London, UK, 1999. Springer-Verlag.
9. Object Management Group. Omg model driven architecture. `http://www.omg.org/mda/`.
10. Daniel González Ignacio Peláez, Francisco Almeida. High level parallel skeletons for dynamic programming. *Parallel Processing Letters*, To appear, 2006.
11. Herbert Kuchen. A skeleton library. In *Euro-Par'02: Proceedings of the 8th Euro-Par Conference on Parallel Processing*, pages 620–629, London, UK, 2002. Springer-Verlag.
12. MathWeb.org. Omdoc: Open mathematical documents. `http://www.mathweb.org/omdoc/`.
13. The OpenMath Society. Openmath. `http://www.openmath.org/`.
14. Dr. Brian Thomas and Dr. Ed Shaya. extensible data format (xdf). `http://xml.gsfc.nasa.gov/XDF/XDF_home.html/`.
15. W3C. Document object model (dom). `http://www.w3.org/DOM/`.
16. W3C. Mathematical markup language (mathml) version 2.0 (second edition). `http://www.w3.org/TR/2003/REC-MathML2-20031021/`.
17. W3C. W3c math home. `http://www.w3.org/Math/`.
18. W3C. The w3c mathml software list. `http://www.w3.org/Math/Software/`.
19. W3C. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl`.
20. W3C. World wide web consortium. `http://www.w3.org/`.

Managing Grid Computations: An ORC-Based Approach*

A. Stewart[1], J. Gabarró[2], M. Clint[1], T. Harmer[1], P. Kilpatrick[1], and R. Perrott[1]

[1] School of Computer Science, The Queen's University of Belfast, Belfast BT7 1NN, Northern Ireland

[2] Dep. LSI, Universitat Politècnica de Catalunya, Jordi Girona, 1-3, Barcelona 08034, Spain

Abstract. In this paper a model of grid computation that supports both heterogeneity and dynamicity is presented. The model presupposes that user sites contain software components awaiting execution on the grid. User sites and grid sites interact by means of managers which control dynamic behaviour. The orchestration language ORC [9,10] offers an abstract means of specifying operations for resource acquisition and execution monitoring while allowing for the possibility of non-responsive hardware. It is demonstrated that ORC is sufficiently expressive to model typical kinds of grid interactions.

Keywords: Grid, ORC, components, constraints, interfaces.

1 Introduction

A computational grid [5] provides an infrastructure for acquiring and utilising computing resources (either hardware or software or both). For the applications programmer, constructing software to exploit fully the available resources presents a considerable challenge. Developing correct and efficient grid applications gives rise to all of the difficulties associated with distributed and/or parallel programming and, in addition, presents challenges arising from the dynamic nature of grid resources: the resources available for computation may vary *during* the lifetime of an application. Thus, in addition to consideration of the functionality of the application and the synchronization issues arising from the distributed/parallel nature of the algorithms, attention must also be paid to the (perhaps) transient nature of the computational resources available and/or communication channels, with appropriate action being taken if adaptation is required.

* This research is carried out under the FP Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265). A preliminary version of this paper was presented at the CoreGRID workshop *Integrated Research in Grid Computing* held at Pisa, 28-30 November, 2005. J. Gabarró is also partially supported by FET proactive Integrated project 15964 (AEOLUS) and by Spanish project TIN2005-09198-C02-02 (ASCE).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 278–291, 2006.
© Springer-Verlag Berlin Heidelberg 2006

For the most part, the development of grid applications currently requires that issues of dynamicity are addressed at a low level by the applications programmer, through calls to grid middleware such as the Globus Toolkit. As a result few grid applications exploit in full the dynamic resource usage envisaged in fully fledged computational grids. There are, however, ongoing attempts to build environments which support the dynamic adaptivity required for the management of grid resources [1,2,11]. The goal of user-friendly grid environments requires the development of grid applications with autonomic (in the sense of [7]) capability to allow for adaptation to changing grid conditions *without* programmer intervention. Such capability is still a long way off, although the extension to ASSIST described in [1] represents a significant early step.

In essence, the challenge for researchers in grid computing is first to devise environments to support hands-on user management of grid applications, and then to provide autonomous management of such applications so that the applications programmer can focus entirely on the functional behaviour required without the additional burden of managing the resources employed. In both cases a clear separation of grid management and functionality provision is desirable. To this end, it is the contention of the authors that there are clear benefits to be had from the development of an abstract model of grid computation which separates the grid management issues from functionality concerns. This paper proposes such an abstract model.

The approach taken defines a component to be the underlying unit of computation. A grid application is modelled as a network of cooperating components. Management of this network is described using the ORC notation of Misra and Cook [10]. ORC is a notation for describing the orchestration of distributed services. It has, as a primitive, the notion of a site call. It provides constructs to orchestrate site calls and thereby a means to describe distributed features such as priorities, time-outs, and failure of sites or communications. It thus supplies the essential ingredients to describe in a precise and concise fashion the operations for the management of grid applications, as distinct from their functionality. In addition, ORC has a well defined semantics [9] which makes possible reasoning about ORC specifications.

The goal of the model proposed here is to bring to grid computation the benefits that formal specification notations such as Z, VDM-SL and LOTOS [6] have brought to sequential and concurrent systems: the facility to describe the essential features of a system devoid of implementation detail; the ability to reason about systems so described using the underlying semantics of the notation; and the possibility of correctness-preserving refinement to implementation. The work described here focuses on the first of these. It is shown that ORC, through its small but powerful set of site combinators, provides the means to describe, in an elegant way, essential features of grid application management including resource discovery, contract negotiation, job placement, progress monitoring and dynamic re-placement.

In §2 some grid-related aspects of a software component are defined. A component may have associated constraints to specify the kinds of hardware required

to execute the component. In order to develop useful, grid-based orchestration expressions a definition of a generic web site is given in §3. In §4 a brief overview of ORC is given. In §5 a basic component manager is specified using ORC.

2 Software Components

In this section an outline of a software component is given. In particular, those aspects of a component that are grid oriented are modelled. Traditionally, a software component may comprise functionality, input data, an interface (defining software dependencies) and an output file (hereafter called an output component). In a grid setting it is additionally necessary to supply *constraints* (which restrict the kinds of hardware that can be used to execute a component). In this paper two different kinds of constraints are defined: *minimum constraints*, which can be used to determine if a grid site offers appropriate resources; and *value constraints*, which can be used to rank grid sites according to their suitability for executing a given component. A component is defined to be a sextuple comprising an external interface (a set of component names, E), an output component name (o), functionality (f), data (d), a minimum constraint (a predicate MC) and a value constraint (an expression VC).

$$component \triangleq (E\ ,\ o\ ,\ f\ ,\ d\ ,\ MC\ ,\ VC)$$

The exernal interface of a component c defines its dependencies such as input data sources and utilised service components. Functionality may be supplied as a combination of program code and service invocations.

Example 1. Consider a user who wishes to execute a FORTRAN program, F, on the grid using local data, d, and a data file, I, supplied by a third party, to produce an output component, R. The user may construct a data component, *indata*, and a program component, FP:

$$\begin{aligned} indata &== (\_, \_, \_, I, \_, \_) \\ FP \quad &== (\{Fortran90Compiler,\ indata\}, R, F, d, MC, VC) \end{aligned}$$

where MC and VC are constraints (see below). The components FP and *indata* may be sent to, and executed on, any site offering the service *Fortran90Compiler* and satisfying the hardware constraint MC. □

2.1 Constraints

Constraints are statements that can be used to specify hardware, performance, cost and network requirements. These have two forms: *minimum* constraints and *value* constraints. A minimum constraint is a set of requirements that *must* be satisfied by any site which is delegated to execute the associated component. A minimum constraint can be modelled abstractly as a predicate.

Example 2. The constraint that a grid machine should have at least 10 processors and a communication link with bandwidth of at least 10^9 bytes/sec may be expressed as a predicate with free variables p and b:

$$MC \triangleq p \geq 10 \wedge b \geq 10^9$$

For a particular site the parameters p and b may be instantiated, allowing the predicate to be evaluated. In order to evaluate a constraint MC on a site s it is necessary to acquire the values of all free variables by means of a dialogue. For example, p might be instantiated for a site, s, by means of a call $s.processors$. The advantage of modelling constraints as predicates is that (uninteresting) underlying dialogue can be excluded. Let s_1 be a site with $p = 4$ and $b = 10^6$ and s_2 be a site with $p = 16$ and $b = 10^9$. Let $MC(s)$ be the value found by instantiating the free variables of MC with the site parameters of s. Then $MC(s_2)$ but $\neg MC(s_1)$. However, consider the weaker constraint MC':

$$MC' \triangleq p \geq 4 \wedge b \geq 10^6$$

Now $MC'(s_1)$ holds (that is, a component with constraint MC' can be executed on s_1 as well as s_2). Note that $\forall g.MC(g) \Rightarrow MC'(g)$ – that is, use of a weaker constraint extends the set of possible sites for placing a component. □

A minimum constraint provides a means of deciding whether or not a site can execute a component. However, it is useful also to be able to determine the"best" site on which to execute a component. A value constraint is an expression with free variables. The expression can be instantiated via a dialogue as above. Instantiation allows a set of potential sites to be ranked according to their ability to meet the value constraint.

Example 3. Consider a component, c, with an associated constraint

$$VC \triangleq 10^{10}/ct + b$$

Here ct is a unit computational cost and b is the available bandwidth. Suppose that the unit costs for s_1 and s_2 are 1 and 1000 respectively. Then $VC(s_1) > VC(s_2)$ – in this case a cheap but less powerful site is preferred. □

3 A Model of a Generic Grid Site

A grid is modelled as a collection of interconnected sites. A user may wish to submit a software component(s) to a grid for execution. To do this a means of navigating the grid (in order to find a suitable site for component placement) is required. In this section a generic definition of grid site is proposed.

A site is defined to be a sextuple comprising a unique name (N), a set of components, or jobs, awaiting execution (C), a collection of services (S) that can be utilised by users, a directory (ID) providing information about grid sites, an

engine (E) which has the potential to execute components to produce results, and a local manager (M) which co-ordinates workflow.

$$gridsite == (N\ ,\ C\ ,\ S\ ,\ ID\ ,\ E\ ,\ M)$$

In a particular site only some of these fields may be instantiated. For example, a "yellow pages" information site may contain only a site name, an information directory and a manager:

$$yellow\_pages = (yp,\ ,\ ,id,\ ,m1)$$

Here id provides information about other sites (for example, the set of sites which offer the capability to execute FORTRAN 90 programs). This information may become obsolete or unreliable after a period of time. The manager $m1$ may interact with other sites and update id as appropriate.

A supercomputer centre offering services, S, (compilers, operating systems, standard libraries such as SCALAPACK etc.) will typically contain a set of jobs awaiting execution, J, a supercomputer, $super1$, for generating results, and a manager. This may be expressed as:

$$super\_computer\_centre = (scc, J, S,\ , super1, m2)$$

Here the manager $m2$ interacts with users, accepts jobs for execution, manages the"queue" of jobs and returns results to users.

Users wishing to submit software artifacts for execution on the grid can themselves be regarded as sites with components and managers.

$$user = (u, C,\ ,\ ,\ ,m3)$$

Here the manager $m3$ is software for placing a set of components C on the grid for execution. An example of such a manager is developed in §5.

3.1 Services

A component that is made available, by a grid site, for general use is called a service. Activation of a service, sv, by a user results in the future execution of sv. Typically, execution will be performed on the local site. However, a grid site may offer services without having a local engine with which to perform computation. In such a situation a site may acquire an appropriate service from a third party. This action is invisible to a user.

The set of services offered by a site may vary with time. User defined components (i.e. software developed by applications programmers) may utilise services through invocation.

Example 4. Consider the following user-defined component:

$$es\ ==\ (\{eigensolver\}, o, eigensolver, d, MC, VC)$$

Here, functionality is provided by invocation of the service $eigensolver$. A site that accepts component es for execution implicitly agrees to the use of its eigensolver service. □

3.2 Information Directories

An information directory provides information about services and hardware offered by grid sites. The information provided by directories allows users to navigate the grid and to extract information which allows components to be placed appropriately. Local site information is considered reliable whereas information about external sites may be obsolete. Directories may provide:

1. *local* site information: for example, the set of local services offered by site s is given by $s.services$

$$s.services :\rightarrow Set(Component)$$

   and the position of component c in the execution queue of s is given by $s.queuepos(c)$.

$$s.queuepos : Component \rightarrow Z$$

   A particularly useful function is $s.can\_execute$,

$$s.can\_execute : Component \rightarrow SiteName \times Z$$

   which takes a component, c, as argument and, if s can execute the component (that is, it can supply the required software interface and meet the hardware constraints), returns a pair comprising the site name s and the result of evaluating the value constraint of c; otherwise it remains silent.
2. *grid wide* information:
   for example a yellow pages site s may be asked to provide the set of all grid sites which can execute a component c ($s.all\_can\_execute(c)$) [1].

$$s.all\_can\_execute : Component \rightarrow Set(SiteName)$$

3.3 Engines and Execution of Components

A set of components C, comprising a program and input data, can be executed on a single site s if the site offers an appropriate range of services and also meets all of the component constraints. A set of components that is accepted for execution (a job) is placed on a queue. An engine is a function which transforms such a set of components into an output component:

$$engine : Set(Component) \rightarrow Component$$

Example 5. Placement of the component set $\{FP,\ indata\}$ (see example 1), by a user u, on an appropriate site s creates a job awaiting execution by the site engine. Execution of the job generates an output component R. R may be returned to u or it may be stored, awaiting collection. □

[1] Note that the information supplied by one site s about another may be out of date.

3.4 Managers

In the definition of grid site above, sets of jobs, sets of available services and information directories are *passive*. A site manager interacts with the grid (and with applications programmers) and controls dynamic behaviour [2]. In this paper a simple user site manager (for placing and monitoring the execution of a component) is identified with an ORC expression. More generally, a grid-site manager controls:

- acceptance of new jobs for execution;
- the range of services currently offered;
- the updating of information directories through searches; and
- the job queue (which may involve placing jobs elsewhere).

A user interacts with a site by means of calls to specific functions (such as *s.all_can_execute* (to navigate) or *s.execute* (to utilise services). In simple situations the manager can decide how to respond using the local site state; however, in general, a manager may also generate calls to third-party sites (with possible side-effects). A user is oblivious to all such secondary communication. This kind of interaction is exactly that supported by the orchestration language ORC.

4 ORC: A Language for Site Orchestration

A brief summary of the language ORC is given here – see [9,10] for a complete description. ORC is based on the idea of a site call. In ORC all operations must be realised as site calls (e.g. there are no in-built arithmetic operations - addition of x and y may be simulated by the site call $add(x, y)$). In general a site call M may update the recipient site which, in turn, may call other sites and reply. A fundamental concept of ORC is that a site call may fail (i.e. the sender may not receive a reply). This may be due to a faulty network (either the outgoing or incoming message may fail) or even to the recipient site being down. There are some special sites:

- 0 never responds (0 can be used to terminate execution of threads);
- if b returns a signal if b is true and remains silent otherwise;
- $RTimer(t)$, always responds after t time units;
- *let* always returns (publishes) its argument.

In addition, calls made to the generic grid site defined in §3 are considered.

ORC site calls may be orchestrated by means of expressions. The simplest expression is a site call, possibly with parameters. More complex expressions can be defined as follows, where E_1 and E_2 are expressions:

[2] In this paper a single manager is specified for an entire grid site. In practice, a site may devolve responsibility to the service level – thus, it may be the case that each service, the information directory and the engine incorporate their own managers.

1. operator $>$ (sequential composition)
   $E_1 > x > E_2(x)$ evaluates E_1, receives a result x, calls E_2 with parameter x. If E_1 produces two results, say x and y, then E_2 is evaluated twice, once with argument x and once with argument y. The abreviation $E_1 \gg E_2$ is used for $E_1 > x > E_2$ when evaluation of E_2 is independent of x.
2. operator $\mid$ (parallel composition)
   $(E_1 \mid E_2)$ evaluates E_1 and E_2 in parallel. Both evaluations may produce replies. Evaluation of the expression returns the merged output streams of E_1 and E_2.
3. where (asymmetric parallel composition)
   E_1 where $x :\in E_2$ begins evaluation of both E_1 and $x :\in E_2$ in parallel. Expression E_1 may name x in some of its site calls. Evaluation of E_1 may proceed until a dependency on x is encountered; evaluation is then delayed. The first value delivered by E_2 is returned in x; evaluation of E_1 can proceed and the thread E_2 is halted.

4.1 Site Failure

In the case of site failure (observed as silence) a user may remain waiting for a response (non-termination). Silence may be temporary – due to a delay in response from the target site – or permanent. Evaluation of some ORC expressions *must* succeed. For example, given a grid site g the expression

$$Terminate \triangleq \mathsf{let}(g) \ \ \mathsf{where}\ g :\in \{N \mid Rtimer(100) \gg let(stop)\}$$

has one thread $Rtimer(100) \gg let(stop)$ comprising local site calls only. These calls must succeed and so evaluation of the expression must terminate. However, this is not the case for an arbitrary site call. Grid programs should react when a response, which they are awaiting, is not forthcoming. Typically, if there is no response to a site call within a specified period then an exception handling routine will be invoked. One way of dealing with site calls that do not respond is to repoll the site. Thus, instead of conducting a single poll at once, polls may be carried out at regularly or irregularly spaced intervals. Two such polls, $Poll^*$ and $TPoll$, which repeatedly instantiate an ORC expression, E, are defined below:

$$Poll^*(E) \triangleq \mathsf{let}(\mathsf{r}) \ \ \mathsf{where}\ r :\in \{ \mid_{t \in \mathcal{N}} Rtimer(t) \gg E\}$$

Here E is instantiated (infinitely often) at regularly spaced intervals. Each thread in $Poll^*$ waits indefinitely for a reply.

In many situations a user may wish to poll repeatedly until a response is received. With the time poll, $TPoll$, multiple threads are generated. However, threads which do not respond after t time units are ignored. In effect, old threads are "cancelled" before new threads are invoked. The arguments f and t may be used in the definition below to vary the interval between polls:

$TPoll(E, t, f) \triangleq$
  if $flag \gg let(s) \mid$ if $\neg flag \gg TPoll(E, f(t), f)$
  where $(flag, s) :\in \{E > s > let(true, s) \mid Rtimer(t) \gg let(false, failure)\}$

Note that only one of the component expressions in the parallel statement
(if $flag \ldots) \mid$ (if $\neg flag \ldots$)
can become active – in this case either $TPoll(E, f(t), f)$ or $let(s)$. Evaluation of $TPoll(E, 1, \lambda x.x)$ generates a sequence of polls separated by 1 time unit while evaluation of $TPoll(E, t, \lambda x.x + x)$ generates a sequence of polls at times 0, t, 2t, 4t, etc. A new thread is activated at time $f(t)$ if the earlier thread fails to respond in the interval $t, \ldots, f(t)$.

5 The Behaviour of a Component Manager

In this section we consider a number of ways by which a component manager on a user site can select, utilise and monitor grid resources.

5.1 Site Selection

Suppose that a user (or manager) knows the constituent sites in a grid $\mathcal{G}$:

$$\mathcal{G} = \{g_1, \ldots, g_n\}$$

Consider the selection of an appropriate site on which to place a component, c, for execution.

Arbitrary selection. Grid site selection can be made by choosing the first suitable site to respond:

$$Select\_First(c, \mathcal{G}) \triangleq \{let(g) \text{ where } (g, v) :\in (\mid_{g_i \in \mathcal{G}} \; g_i.can\_execute(c))\}$$

This selection generates a call to each of the sites in $\mathcal{G}$ to determine which are suitable for the placement of c; the first site to respond returns its grid site name, g, and the value, $VC(g)$. Evaluation of $Select\_First$ publishes g. Note that if none of the sites can execute c then the evaluation will not terminate.

Selection using a site ranking operation. The first site to respond may not be the best location for placing c. The set of best sites on which to place c, as determined by the value constraint VC, is:

$$\{b \in \mathcal{G} \mid \forall g \in \mathcal{G}.VC(b) \geq VC(g)\}$$

Here all sites which maximise the constraint VC are included in the set. The ranking of a particular site, s, may be found using the operation call $s.can\_execute(c)$. The result of such a call is a pair comprising the site name and its constraint value; these results are passed to a local variable z by means

of the operation *pass*. The expression *Rank*, below, constructs multiple threads which send a stream of site information to variable z; after calling *pass* each thread is terminated.

$$Rank(c, \mathcal{G}) \triangleq z.null \gg \\ (\,|_{g_i \in \mathcal{G}}\ g_i.can\_execute(c) > (g, v) > z.pass(g, v) \gg 0)$$

Here z is initialised by means of the site call *null*. The best site available after t time units may be selected as follows:

$$Top\_Rank(c, \mathcal{G}, t) \triangleq let(g) \text{ where} \\ (g, v) :\in (Rank(c, \mathcal{G}) \,|\, Rtimer(t) \gg z.highest)$$

The pair which has the highest constraint value may be retrieved by the call $z.highest$. Evaluation of Top_Rank is guaranteed to terminate; however, when none of the active sites in $\mathcal{G}$ can execute c, a null site name is published.

Variants of this operation may be applied to rank sites using different metrics. For example, suppose that yellow pages directories provide site reliability information (the probability of a site responding, say). Then a ranking operation could be constructed using reliability information.

Enlarged selection using a trawling operation. A user who polls the set of sites $\mathcal{G}$ and does not receive in reply a valid site name may wish either to poll a larger set of sites or alter the given constraint set. A larger set of potential sites may be found using grid information directories.

$$Trawl(c, \mathcal{G}) \triangleq x.empty \gg \\ (\,|_{g_i \in \mathcal{G}}\quad g_i.all\_can\_execute(c) > \mathcal{G}' > x.union(\mathcal{G}') \gg 0)$$

Here each thread in $Trawl$ determines a set of sites having the potential to execute c. The sets are combined by distributed union and the result is held in a local variable, x. Distributed union is realised through a stream of *local* site calls, *union*, each of which has the side effect of extending x with $\mathcal{G}'$. The site call *empty* is used to initialise x. After t time units the set currently stored by x is extracted using the operation *get*:

$$Trawl\_Select(c, \mathcal{G}, t) \triangleq Top\_Rank(c, \mathcal{G}', t) \\ \text{where } \mathcal{G}' :\in (Trawl(c, \mathcal{G}) \,|\, Rtimer(t) \gg x.get)$$

Note that the set of sites $\mathcal{G}'$ found by browsing information directories may not be valid. Direct contact is made with each of these sites using the expression $Top\_Rank(c, \mathcal{G}')$ to verify that it can be used to place c.

Selection with weakened value constraints. A user may, in the event of not being able to find a suitable site to place c, weaken the associated component value constraint. Let c' be a component which is the same as c except that it has a

weakened value constraint. A selection mechanism which first tries to find where to place c and, if unsuccessful, then tries to find where to place c' is:

$$\begin{aligned} &Weakened\_Select(c, \mathcal{G}, t) \triangleq \\ &\quad (\mathsf{if}\ \neg\mathsf{null}(g) \gg let(g) \,|\, \mathsf{if}\ \mathsf{null}(g) \gg Trawl\_Select(c', \mathcal{G}, t)) \\ &\qquad \mathsf{where}\ (g, v) :\in Trawl\_Select(c, \mathcal{G}, t) \end{aligned}$$

This strategy can be repeated to allow the constraint to be further weakened.

It may happen that a user tries to select a site to place a component when the grid network is congested – in such circumstances responses to site calls may not be delivered. To make site selection more robust, it may be desirable to use a form of repeated polling (as in §4.1).

5.2 Component Placement

Assume that the evaluation of an ORC expression $Select(c, \mathcal{G})$ returns a site, in grid $\mathcal{G}$, that can be used to execute the component c (if such a site exists) and remains silent otherwise. It remains for the user to contact the designated site (or sites in the case of a distributed job) and to establish a contract for the execution of c. Assume that a call $g.execute(c)$, if successful, returns the output component generated through execution of c on g. Note that a significant delay may occur between invocation of *execute* and receipt of the output component. In the simplest case a component is scheduled for execution on a single grid site. However, distributed placement may occur for a set of components. A variety of techniques for the placement of components is considered below.

Single site placement. A single site job placement by user u may be made as follows:

$$Place(\{c\}, \mathcal{G}) \triangleq Select(c, \mathcal{G}) > g > g.execute(c)$$

A user may execute a component and publish the output by evaluating the expression:

$$Place(\{c\}, \mathcal{G}) > f > let(f)$$

Place is not robust and may fail if g does not agree to the proposed placement. It can be made more robust using techniques similar to those used in §4.1.

Multiple site placement. Two independent components c_1 and c_2 can be placed and executed independently as follows:

$$Independent(\{c_1, c_2\}, \mathcal{G}) \triangleq (\ Place(\{c_1\}, \mathcal{G}) \,|\, Place(\{c_2\}, \mathcal{G}))$$

Output can be recovered in a way similar to that described above. More interesting is the situation where a distributed computation contains a dependency. Consider two components c_1 and c_2 where c_1 has output component name f (i.e. an output file) and c_2 has an external (input) interface $\{f\}$. One method of

placing c_1 and c_2 is first to place c_1, await the return of the component f, and then place a job comprising c_2 and f:

$$Hub(\{c_1, c_2\}, \mathcal{G}) \triangleq Place(\{c_1\}, \mathcal{G}) > f > Place(\{c_2, f\}, \mathcal{G}))$$

Here the user's site acts as a hub for controlling the orchestration (i.e. the data file is returned to the user and then forwarded for subsequent placement). An alternative placement approach, which passes the intermediate result f directly between the two selected grid sites, is:

$$NoHub(\{c_1, c_2\}, \mathcal{G}) \triangleq Place(\{c_2, f\}, \mathcal{G}) \text{ where } f :\in (\ Place(\{c_1\}, \mathcal{G})\ )$$

Evaluation of $NoHub$ tries to place two jobs simultaneously. However, there is a dependency, f, between the two limbs of the where clause. Thus, in this case, evaluation of the asymmetric composition is realised as a sequential composition. However, it is possible to carry out some of the orchestration in parallel by selecting a site for c_2 before introducing the dependency f:

$$\begin{aligned} ReserveFirst(\{c_1, c_2\}, \mathcal{G}) \triangleq\ & Select(c_2, \mathcal{G}) > g > g.execute(\{c_2, f\}) \\ & \text{where } f :\in Place(\{c_1\}, \mathcal{G}) \end{aligned}$$

Here, selection can be performed without reference to f (since, in this case, f is simply a data file and so does not influence site selection).

5.3 Monitoring Remote Execution

Consider again a single site placement where a component c has been placed on a grid site g by a user. A delay may ensue before computing resources become available for executing c. A user may remotely monitor the position, on a queue, of a job with component c; if progress is not satisfactory the component may be moved to another site. Suppose that c is at position n in a queue (the value n may be determined by an earlier call. $g.queuepos(c)$) and that the queue is to be monitored every t time units. Then

$$\begin{aligned} Monitor(c, g, n, t) \triangleq\ & g.queuepos(c) > m > \\ (\ & \text{if } (m \leq 2) \gg 0 \mid \\ & \text{if } (2 < m \wedge m < n) \gg Rtimer(t) \gg Monitor(c, g, m, t) \mid \\ & \text{if } (2 < m \wedge m \geq n) \gg let(false, reschedule)\ ) \end{aligned}$$

Evaluation of this expression terminates silently if c is near the head of the execution queue. However, if c is further down the queue progress is monitored after every t time units; if progress is not satisfactory (that is if the position of c in the queue has not decreased) then a signal to reschedule the computation is published. Again techniques similar to those used in §4.1 can be used to make $Monitor$ more robust.

A manager for controlling the placement of a component can be defined as follows:

$$Manage(c, \mathcal{G}) \triangleq Select(c, \mathcal{G}) > g > Control(g, c)$$

Evaluation of *Control* generates two threads: an execution thread and a monitoring thread. If the behaviour of site g is satisfactory then monitoring terminates silently; otherwise the monitor publishes a $false$ flag and the execution thread is terminated. A single *Reschedule* thread is now invoked:

$$Control(g,c) \triangleq \mathsf{if}\ flag \gg let(f) \,|\, \mathsf{if}\ \neg flag \gg ReSchedule(c,g,\mathcal{G}\setminus\{g\})$$
$$\mathsf{where}\ (flag,f) :\in$$
$$\{g.execute(c) > f > let(true,f) \,|\, Rtimer(t) \gg Monitor(c,g,n,t)\}$$

Here n and t are user-defined parameters which control the behaviour of the monitor. *ReSchedule* involves the invocation of a cancel operation (on site g) and the re-invocation of selection and placement services:

$$ReSchedule(c,g,\mathcal{H}) \triangleq (\ g.cancel(c) \gg 0 \ |\ Place(\{c\},\mathcal{H})\ )$$

6 Discussion

In this paper a description of a component manager which selects and utilises grid sites for component execution is presented. In the case of excessive delay at a grid site the manager can reschedule its computation elsewhere. However, the behaviour of orchestrations, for certain types of applications, needs to be more complex. For example,

- The ORC model described here assumes that site calls are independent. The case of grid computations involving several components which must *synchronise* during execution is not considered here. One way to represent synchronisation is to extend the model by the addition of a special site *syn* which accepts input from several sources and, when all inputs have been received, returns a result to all of the sources. Work on this extended model is in progress.
- A distributed computation might be orchestrated by booking several execution services at some common fixed time – negotiations to determine a suitable time could be orchestrated in a way similar to that used in the common meeting example in [10].
- The expression *Manage* can be used to reschedule jobs that are currently placed on an execution queue. It may be worthwhile to monitor execution and, in the case of slow progress, to transfer the component and its execution environment to another site – such kinds of transfer are considered in the ambient calculus [4].

It should be noted that ORC is suitable for describing other kinds of grid manager. For example, the component manager *Manage* is an ORC expression which is evaluated remotely in order to control the placement of a component c. However, another possibility is to make the component *autonomic* by the inclusion of an *internal* ORC expression whose evaluation will control the movement

of c. Thus, a user manager may carry out an initial component placement. Thereafter, the internal ORC expression may be evaluated to monitor progress and to move the component if necessary.

The novel features of grid computing are dynamism and uncertainty. ORC provides the basic materials for describing operations in such a world – for example, the ability to perform time-outs and to cancel the execution of threads. Alternative approaches for defining web-based computations include the Pi-calculus [8] and service combinators [3]. In this paper it is demonstrated that the expressive power of ORC is well suited to the specification of dynamic behaviour. In an extension of the work reported upon here it is intended to show how specifications of the sort presented here may be subjected to formal reasoning.

References

1. Aldinucci, M., Petrocelli, A., Pistoletti, E., Torquati, M., Vanneschi, M., Veraldi, L., Zoccolo, C.: *Dynamic reconfiguration of grid-aware applications in ASSIST.* In J. C. Cunha and P. D. Medeiros, editors, *Proc. of 11th Intl. Euro-Par 2005: Parallel and Distributed Computing.* LNCS 3648, pages 771–781, 2005.
2. Buisson, J., Andre, F., Pazat. J-L.: *Dynamic adaptation for grid computing.* In P.M.A. Sloot, A.G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005 (European Grid Conference, Amsterdam, February 14-16, 2005, Revised Selected Papers)*, LNCS 3470, pages 538–547, Amsterdam, Springer-Verlag, June 2005.
3. Cardelli, L; Davis, R.: Service Combinators for Web Computing. IEEE Transactions on Software Engineering, Vol 25, No 3, May/June pages 309–316, 1999.
4. Cardelli, L; Gordon, A: Mobile Ambients. In *Foundations of Software Science and Computation Structures, FOSSACS'98.* LNCS 1378, pages 140–155, 1998.
5. Foster, I., Kesselman, C.: *The GRID: Blueprint for a new computer infrastructure.* Morgan Kaufmann, 1999.
6. Gaudel, M-C.: *Formal Specification Techniques (Extended Abstract).* In B. Fadini, L. Osterweil, A. van Lamsweerde, editors, *Proceedings of the 16th international conference on Software engineering*, pages 223–227, Sorrento, Italy, IEEE Computer Society Press, May 1994.
7. Kephart, J.O., Chess., D.M.: *The Vision of Autonomic Computing* In *IEEE Computer* Vol. 26, No. 1, pages 41–50, 2003.
8. Milner, R.: *Communicating and mobile systems: the π-calculus.* Cambridge University Press, 1999.
9. Hoare, T., Menzel, G., Misra J.: A Tree Semantics of an Orchestration Language. In M. Broy, editor, *Proc. of the NATO Advanced Study Institute, Engineering Theories of Software Intensive Systems*, NATO ASI Series Marktoberdorf, Germany, 2004.
10. Misra J.; Cook, R.: Computation Orchestration: A Basis for Wide-Area Computing To appear in Journal of Software and Systems Modeling. A preliminary version of this paper appeared in *Proc. of the NATO Advanced Study Institute, Engineering Theories of Software Intensive Systems*, NATO ASI Series Marktoberdorf, Germany, 2004.
11. Sathish Vadhiyar, S., Dongarra, J.: *Self Adaptability in Grid Computing* In *Concurrency and Computation: Practice and Experience, Special Issue: Grid Performance*, Vol. 17, No. 2–4, pages 235–257, 2005.

Adaptive Technique for Automatic Communication Access Pattern Discovery Applied to Data Prefetching in Distributed Applications Using Neural Networks and Stochastic Models*

Evgueni Dodonov[1], Rodrigo Fernandes de Mello[1], and Laurence Tianruo Yang[2]

[1] Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
São Carlos, SP, Brazil
[2] St. Francis Xavier University, Antigonish, NS, Canada
lyang@stfx.ca

Abstract. The distributed computing performance is usually limited by the data transfer rate and access latency. Techniques such as data caching and prefetching were developed to overcome this limitation. However, such techniques require the knowledge of application behavior in order to be effective. In this sense, we propose new application communication behavior discovery techniques that, by classifying and analyzing application access patterns, is able to predict future application data accesses. The proposed techniques use stochastic methods for application state change prediction and neural networks for access pattern discovery based on execution history, and is evaluated using the NAS Parallel Benchmark suite.

1 Introduction

The evolution of general purpose machines and computer networks have influenced the developing of the distributed computing area for the past decades. However, the distributed computing performance is usually limited by the data transfer rate and access latency. While computational power have increased in several orders of magnitude over last years, the slow improvements in network transfer rates have been limiting the overall performance of distributed applications. This has motivated the development of several techniques to overcome this limitation.

Among such techniques it is possible to mention data caching and prefetching [1]. Both techniques are addressed to reduce data access latency. The first makes this reduction by using local memory for data storage and the second anticipates data accesses, transferring data while network is idle.

In order to apply these techniques it is necessary to know the application behavior, in this case, represented by data access patterns. Some access patterns can be discovered using discrete observation [1], statistical [2] or analytical [3] algorithms. Although, complex applications require different strategies.

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 292–303, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In this scope, we propose a new access pattern discovery and analysis techniques based on neural networks and statistical approaches. In order to apply these techniques, we have to extract and classify the application communication behavior. For this, the first phase is responsible for monitoring application execution by using an analyzing tool based on the approach introduced in the *GridBox* project [4]. The obtained application communication behavior is classified using an adaptive, unsupervised and on-line neural network [5].

The classified application behavior is used in two different and independent data access prediction techniques: the first creates a Markov chain by defining arcs among the classified states, according to the application communication variation during its execution; and the second builds a time series, following the application communication state changes, which is submitted to a prediction neural network capable of defining future communication behavior. By both techniques we may infer when and how much data should be accessed, minimizing the total application execution time. The proposed method is evaluated using NPB 3.1 [6] benchmark suite.

This paper is organized as following: section 2 overviews the related work and technologies used in this work. The proposed architecture is shown in section 3, and the experiment results are discussed in section 4. Finally, section 5 summarizes the obtained results and concludes this paper.

2 Related Work

The performance of storage devices is considerably inferior when compared with the memory speed. This is mostly observed in distributed applications which are limited by network throughput and data access time. In this context, techniques have been developed for caching and prefetching which provide a significant application performance gain. An efficient prefetching strategy allows the most relevant data to be stored in memory prior to application requests, reducing the latency accesses and increasing the performance of I/O operations.

In order to support anticipated data requests, the file system must know *a priori* which data an application will need next. In order to discover future data blocks, the application access pattern – read and/or write operations – must be known [1]. An access pattern is highly application and system dependent, and can vary from simple sequential accesses to highly complex and apparently random access sequences.

Fundamental prefetching concepts are presented in [1,7], discussing the prefetching techniques and their integration with caching mechanisms. Strategies for sequential access pattern discovery are also proposed in [8], introducing a framework for data prefetching based on the probability of future I/O operations. The work present the *CPS* prefetching algorithm which uses statistical probabilities of I/O operations. The proposed framework supports pluggable access pattern discovery algorithms, making possible to extend *CPS* with different data access pattern prediction strategies. Prefetching implementation for distributed systems are also discussed in [9,10].

Prefetching strategies for non-sequential data patterns, such as used in distributed shared memory systems, are discussed in [11]. Complex data prefetching strategies can

be determined using semantic structures and analytical approaches [3], data accesses classification [12], and stochastic approaches such as hidden Markov models [2].

Several works relate that the artificial intelligence techniques such as the artificial neural networks has the potential to improve the pattern discovery and prediction [5,13] efficiency. This has motivated this work to evaluate such techniques for classification and prediction of distributed applications behavior. The adopted neural networks were the *ART 2A* network [14] and the *TDNN* network [15].

ART 2A network is a part of the Adaptive Resonance Theory (ART) neural network architecture developed by Grossberg [14]. The basic ART system is an unsupervised learning model and typically consists of neuron layers for comparison and recognition, a vigilance parameter, and a reset module. The vigilance parameter, named ρ, has considerable influence in the classification: the higher is the vigilance parameters, the more accurate is the classification. Although, the accuracy implies in lost of data generalization. This work has adopted the *ART 2A* network due to its ability to classify a series of patterns into a number of different clusters in a controllable way.

Time delay networks (TDNN), introduced by Waibel [15], are a group of neural networks that have a special topology. They are used for position independent recognition of features within a larger pattern. In a traditional neural network the basic unit computes the weighted sum of its inputs and then forwards it through a nonlinear function to other units. In TDNN, the basic unit is modified by introducing n delays to the input, so a input layer composed of y inputs would generate $z = y \cdot (n + 1)$ inputs to the network. Introducing the delays the neural network can relate and compare the current input to data previously observed, in this way it effectively implements a short-term memory mechanism [15]. The output values are compared to the expected ones and the error value is obtained and backpropagated through the network, updating the weights and decreasing the prediction error. This procedure is repeated until the results converge to the expected outputs. The TDNN network ability to learn the data behavior has motivated this work to adopt it to make predictions based on historical informations.

3 Model for Discovery and Prediction of Application Data Access Patterns

We propose a new access pattern discovery and analysis techniques based on neural networks and statistical approaches. The technique is based on transparent and online application behavior extraction, which is based on the *GridBox* project approach [4]. The obtained application communication behavior is classified using the *ART 2A* neural network in an adaptive and unsupervised manner [5] to reduce the dimensionality of data.

The classified application behavior is used in two different and independent data prefetching techniques: the first creates a stochastic application behavior prediction model according to application communication variations during the execution using Markov chains; and the second builds a time series, representing the application communication state changes, which is submitted to a prediction neural network capable of defining future communication behavior. Both techniques allow to determine when and

how much data should be accessed, minimizing the data access latency. The proposed method is evaluated using NPB 3.1 [6] suite.

This work is composed of the following steps:

1. **On-line and continuous application communication behavior extraction**
   The first phase of this work consisted of automatic extraction of application communication behavior during its execution. In order to do so, an automated profiling suite was developed, based on the *GridBox* project [4]. This suite was specifically tuned to monitor and register all communication operations, requiring no application source code modification, registering the following data:
   - Sequence of operations during execution, allowing to restore the application behavior afterwards;
   - CPU time, between consecutive communication operations, used to determine the application processing cost at any given point of execution;
   - Location and size of modified data for each distributed operation, which is used to determine and predict the application access patterns;
   - I/O, memory and network usage for each distributed operation, which could be used to improve the application scheduling and load balancing for different heterogeneous networks.

   With this data it is possible to reconstruct application behavior during the whole course of execution. This information is used to classify application communication behavior patterns as shown next.
2. **Application behavior classification**
   Having the application execution trace collected, the application behavior is classified using the *ART-2A* [5] neural network. The pattern classification is performed analyzing the sequence of application data accesses which are grouped into clusters of similar behavior.

   During this operation, the vigilance parameter ρ varies from $0.7 \text{to} 0.9999999999$. This helps to find the most adequate classification accuracy, which is determined when the inter-cluster and intra-cluster distance intercept each other, as shown in [5].

   Having the best vigilance parameter, the application communication behavior is classified into clusters. Each cluster represents application functionalities in different periods of executions, for example, periods of high, medium and low data exchanges among the tasks of the same distributed application. The transcription data is generated, describing the sequence of application operations in a form of cluster transitions. For example, the application is transferring a high volume of data at the instant t_0 (e.g. classified in the cluster 0), at the next instant t_1, it stops communicating (e.g. cluster 1) to perform a CPU-bound operation, and at the instant t_2, it responds with obtained results with a low network utilization (e.g. cluster 2). The sequence of application transitions is used in two different and independent techniques as shown next.
3. **Application behavior prediction**
   We propose two different prediction techniques based on the application behavior extraction presented before. The first one calculates the incidence matrix to

generate Markov Chains for a statistical approach, allowing to define the next state based on the current one. The second transforms the classified behavior into a time series for prediction using TDNN neural network. This method allows to predict a sequence of future application state changes based on application execution history.

The first model uses an incidence matrix that contains all application state transitions, calculating the probability of each state transition. This allows to predict all states that can be reached by the application at any given execution point and the probability of accessing them. This model does not consider the application execution history and gives stochastic results which could be applied by prefetching algorithms such as CPS [8].

Besides applying the Markov Chains method, the application behavior is also evaluated using the TDNN [15] neural network. The classified application behavior is transformed into a time series, showing how the application behavior varies during execution. This time series is structured as a set of vectors of size n (formally defined as lag [15]), such as $v_0 = e_0, e_1, ..., e_{n-1}$ where e_k is a communication pattern classified by *ART 2A*, to predict the next point e_n. This organization is similar to a sliding-window, where the next vector is $v_1 = e_1, ..., e_n$, used to predict e_{n+1} and so on. These input vectors are used to train a n-lag TDNN neural network for predicting future data accesses.

This model uses application execution history (represented as a time series of application state changes) as basis for application behavior prediction. Differently from the statistical model, this allows to efficiently predict a set of consecutive application state transitions, thus allowing a larger prefetching window. In this way, it is also possible to detect application behavior changes, adapting the prediction as necessary.

4 Evaluation Results

The NAS Parallel Benchmark (NPB) was used as testbed for this work, as it is widely employed for parallel and distributed application evaluations. The suite is composed by a set of benchmarks (named *ep, mg, cg, ft, is, lu, sp* and *bt*) , which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications. Each of these benchmarks can be compiled into different classes (A, B, C, W and S), each offering different application loads, such as memory and cpu usage, execution time, problem size, etc. Most of the benchmarks are written in FORTRAN, with the exception of the *IS* that is programmed in C. In this work, the NPB-MPI suite was adopted using LAM-MPI [16] MPI implementation.

While the proposed architecture allows on-the-fly application evaluation and behavior prediction, for didactic reasons the NPB benchmarks were fully executed and their results analyzed after executing. For application prediction, execution traces were splitted, using the first 50% of execution for training the TDNN neural network. The prediction was performed using the remaining 50%, analyzing the prediction error.

IS.A benchmark:

```
# MPIDEBUG: GridBox log file.
# Data format:
# <ST> <UT> <PF> <PR> <BI> <BO> <VAR> <IN> <OUT> <SIZE>

60000 1600000 383 8278 0 0 -1 -1 -1 -1
0 0 0 0 0 0 184868512 1 0 1029
0 0 0 0 0 0 184860224 0 1 1029
0 0 3 0 0 0 -1 -1 -1 -1
0 0 0 0 0 0 184872640 1 0 1
0 0 0 0 0 0 184882912 0 1 1
...
```

Fig. 1. Execution trace for **IS.A** benchmark

Due to the page limitation, this paper does not include all obtained results. Thus, only the results for the *IS.A* and *CG.A* benchmarks are discussed here, having the results for other benchmark freely available at the project site [1].

4.1 Application Behavior Extraction

In order to determine the application behavior, an application tracer was developed, based on the GridBox [4] project. The application behavior was monitored, saving application communication operations. In case of MPI applications, such points are delimited by MPI function calls. The application tracer intercepted MPI calls, saving application usage statistics at each interruption point.

In order to monitor the NPB, it was compiled without modifications, and executed inside the GridBox environment. During the execution, application behavior was monitored and saved into an independent file for each process (running on different machines) for posterior analysis.

The information captured is shown in figure 4.1, with **ST** being *System time*, **UT** – *User Time*, **PF** – *Page Faults*, **PR** – *Page Reclaims*, **BI** – *I/O Blocks In*, **BO** – *I/O Blocks Out*, **VAR** – *Address of accessed variable*, **IN** – *Flag to indicate that the variable was read*, **OUT** – *Flag to indicate that the variable was written* and **SIZE** – *Size of accessed data*.

As it is possible to see in figure 4.1, periods of CPU usage are followed by data synchronization (communication) among nodes. This allows to determine data access patterns which can be further used to predict application behavior.

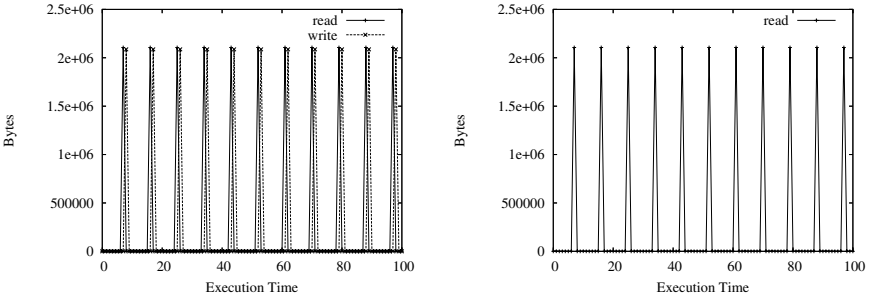
The IS.A benchmark has performed 108 communication operations while CG.A has executed 6724. Thus, in order to make a more didactical comparison of two benchmarks, all charts related to the CG were limited to the first 100 communication operations, once when they are fully-plotted the visualization is jeopardized [2].

4.2 Application Access Pattern

The obtained application trace is used to determine access patterns. This can be performed by selecting the last three fields from the captured data (shown in figure 4.1),

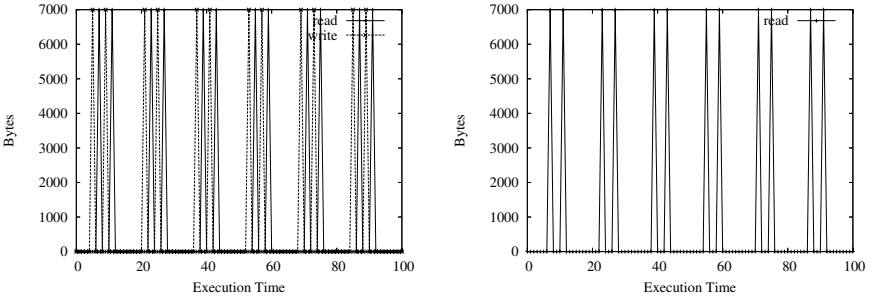
[1] http://www.icmc.usp.br/~mello

[2] The full traces of NPB are available at http://www.icmc.usp.br/~mello



(a) Full access pattern **(b) Read access pattern**

Fig. 2. IS.A access pattern



(a) Full access pattern **(b) Read access pattern**

Fig. 3. CG.A access pattern

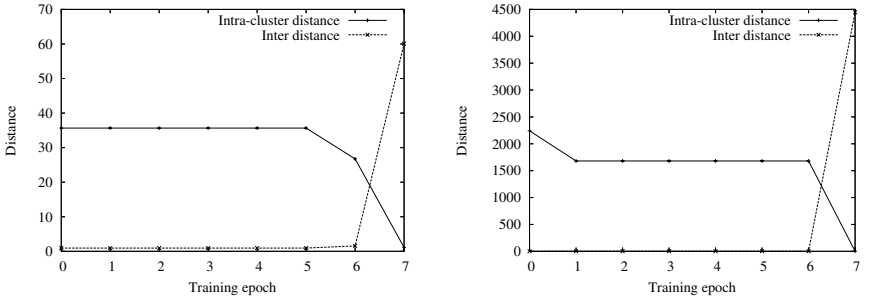
that represents the amount and direction of the I/O operation. The application access pattern can be observed in figure 2.

By using the data from figures 2 and 3, it is possible to observe the behavior of both IS and CG [3] applications. While IS has relatively few CPU operations, synchronized at MPI data exchange routines, CG constantly exchanges small amounts of data over the network. From this data, it is observed that applications whose behavior is similar to IS benchmark are CPU-bound, while the performance of applications similar to CG is limited by network latencies due to the high number of communication operations.

The application behavior and network usage can be applied to determine the most adequate protocol and message sizes according to the network environment, such as LAN, MAN or WAN. As the network latency and transfer rate are highly dependent on the environment, the data transfer approach (such as packet size, protocol, network topology, etc) is highly important to obtain best application performance [17].

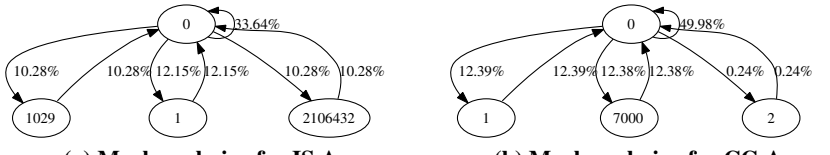
For example, it is observed that applications similar to IS are more likely to behave better in wide area networks (such as grid) than those similar to CG. Thus, combining

[3] As the CG benchmark is heavily IO-bound, the results were limited to first 100 execution points for visualization reasons.



(a) IS.A benchmark **(b) CG.A benchmark**

Fig. 4. Pattern classification threshold discovery



(a) Markov chains for IS.A **(b) Markov chains for CG.A**

Fig. 5. Markov chains

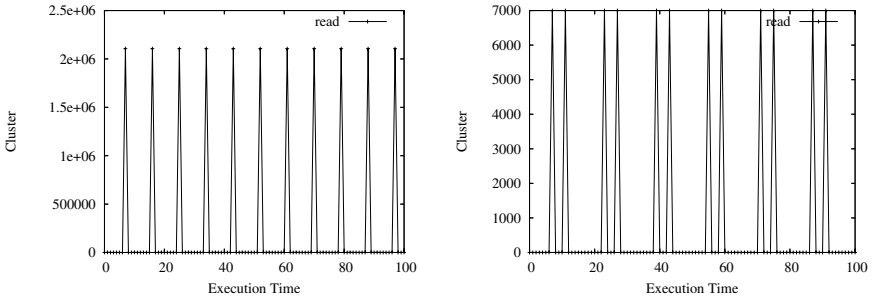
the application behavior data and prefetching algorithms to the knowledge of network environment characteristics, it is possible to optimize the I/O request in a way to provide best overall performance.

4.3 Access Pattern Classification

Having discovered the sequence of application communication accesses, they are classified into clusters with similar characteristics. Thus, all similar operations are grouped into the same cluster.

In order to perform the classification, the ART-2A neural network is used. The sequence of operations is processed by the neural network, generating clusters, and calculating the inter and intra-cluster distances to determine the most adequate ART-2A parameters [5]. The classification continues until the inter and intra-cluster distances intercept each other, as shown in figures 4.a and 4.b. At that point, clusters describe an efficient representation of the different application states as presented in [5].

Having discovered the number of clusters, it is determined the average volume of transferred data of each one using the cluster centroid value. This value represents each cluster mass center and it is influenced by the values of its patterns. In this way, the centroid represents how much data is expected to be transferred over network at each point of application execution. Using this data, it is possible to construct the Markov chain to describe the statistical model of application behavior, as shown in figures 5.a and 5.b.



(a) IS.A benchmark **(b) CG.A benchmark**

Fig. 6. The time series for NAS benchmark

As presented in figures 5.a and 5.b, the transition from each application state to any other can occur with a given probability. This data can be further used to parameterize statistical prefetching algorithms, such as *CPS* [8].

For easier data representation, cluster numbers were used instead of centroid values here. Thus, for example, CPU-bound application state became *state 0*, low network utilization state became *state 1*, high network utilization state became *state 2*, and so on.

4.4 Application Behavior Prediction

Having evaluated and classified application access patterns, the TDNN neural network is used to predict the sequence of future application operations. In this work, the SNNS [18] implementation of the TDNN network was used.

In order to use the TDNN neural network, series of application access patterns were created, specifying what action is expected for each set of application state changes. This is accomplished by creating time series of application behavior, as shown in figure 4.4. By this figure, it is observed the behavior of IS.A and CG.A benchmarks, represented by state changes, where each state is an ART-2A cluster characterized by similar communication behavior.

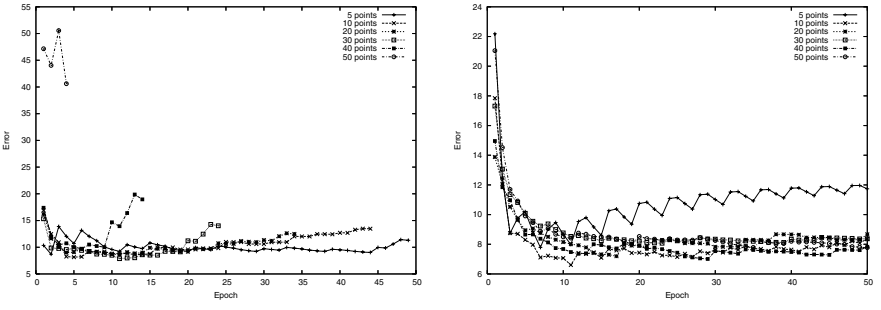
The input vectors, which represent the sequence of application state transitions, are normalized to values between 0 and 1. This normalization speeds up the training phase by providing small variations in the neural network weights. Having this values, series of application behavior patterns are created, as shown in table 1.

In order to predict the future application behavior, the sequence of application communication operations is transcribed into a set of access patterns with lag values of 5, 10, 20, 30, 40 and 50 accesses. This data were used to train the neural network. TDNN networks were created for each lag value and validated against the expected state transitions. This validation provided the error charts presented in figures 7(a,b).

The results shown in figures 7(a,b) were calculated validating the TDNN network trained with the first 50% of application execution patterns (training set) against the remaining 50% (validation set). As can be observed in figures 7(a,b), the best results for both benchmarks are obtained using lag values between 5 and 10, due to small

Table 1. *NPB* access patterns, with $lag = 5$, predicting 1 future pattern

	IS.A	**CG.A**
Number of patterns	48	3356
Detected access pattern	$0 \Rightarrow 0.25 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.5$	$0 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.25 \Rightarrow 0$
Predicted next input	0	0
Detected access pattern	$0.25 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.5 \Rightarrow 0$	$0 \Rightarrow 0 \Rightarrow 0.25 \Rightarrow 0 \Rightarrow 0$
Predicted next input	0	0
Detected access pattern	$0 \Rightarrow 0 \Rightarrow 0.5 \Rightarrow 0 \Rightarrow 0$	$0 \Rightarrow 0.25 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0$
Predicted next input	0.75	0.5
Detected access pattern	$0 \Rightarrow 0.5 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.75$	$0.25 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.5$
Predicted next input	0	0
Detected access pattern	$0.5 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.75 \Rightarrow 0$	$0 \Rightarrow 0 \Rightarrow 0 \Rightarrow 0.5 \Rightarrow 0$
Predicted next input	0	0



(a) IS.A **(b) CG.A**

Fig. 7. TDNN network validation

number of application execution states. As shown in table 1, the IS benchmark does not has enough state transitions to justify the usage of large lag values, thus resulting into few and imprecise results for lag values superior to 30, as can be seen in figure 7.a. This is not the case for the CG benchmark which has much more state transitions.

4.5 Prefetching Functionality

After predicting application state changes, it is possible to define the prefetching mechanism. In this section we illustrate the prefetching algorithm for the IS.A application.

Statistical analysis-based prediction using Markov Chains. This prefetching algorithm is able to predict future application requests based on current application execution state. As observed in figure 5.a, when application is at state 0, with I/O usage of 0 (defined by the centroid value of cluster 0), the following future application actions are possible:

- **10.28% probability** state transition to state 1 with I/O usage of 1029.
- **33.64% probability** to remain at state 0 with I/O usage of 0.

- **12.15% probability** state transition to state 2 with I/O usage of 1.
- **10.28% probability** state transition to state 3 with I/O usage of 2106432.

These values can be used as input to prefetching algorithms such as CPS which may infer the next state based on the current one. In this way they can read-ahead data before the state transition and offer it to the requesting application.

Pattern analysis-based prediction. In this section we consider a prefetching algorithm for IS.A application, using TDNN with a lag of 5 states. As observed in table 1, when application executes a set of transitions from state 0 to states 0.25, 0, 0 and 0.5, the expected next state is 0 (All cluster numbers were normalized between 0 and 1 for the TDNN network. In this way a state 0.25 represents a certain cluster of ART-2A neural network.). Converting the state values back to the centroid values, obtained during the application behavior classification, we may instruct the prefetching algorithm to expect the following behavior[4]:

1. Application performs CPU-based operations (application state 0), defined by I/O usage of 0 (the centroid value of respective cluster 0).
2. Application performs a synchronization operation (application state 0.25), defined by I/O usage of 1 (the centroid value of respective cluster 1).
3. Application performs CPU-based operations (application state 0), defined by I/O usage of 0 (the centroid value of respective cluster 0).
4. Application performs CPU-based operations (application state 0), defined by I/O usage of 0 (the centroid value of respective cluster 0).
5. Application performs an I/O operations (application state 0.5), defined by I/O usage of 1029 (the centroid value of respective cluster 2).

When such sequence of state transitions is encountered, the prefetching algorithm can predict the next application operation with probability of $100 - 11 = 89\%$ to be:

1. Application performs a CPU-based operation (application state 0), defined by I/O usage of 0 (the centroid value of respective cluster 0).

5 Conclusions

In this work we have evaluated the behavior of data accesses for distributed applications, proposing two data prefetching strategies. The first strategy is based on a statistical approach which uses Markov Chains to determine the next application request based on current application execution state. The second one uses TDNN neural network to predict future application accesses based on execution history. The techniques were evaluated using the NAS Parallel Benchmark.

The evaluation results demonstrate that the usage of both statistic-based and application history-based prefetching mechanisms provide high efficiency in discovering distributed data access patterns.

[4] Notice there is a state just to represent the environment without communication in which the application may be processing.

References

1. Kotz, D., Ellis, C.S.: Practical prefetching techniques for multiprocessor file systems. Journal of Distributed and Parallel Databases **1**(1) (1993) 33–51
2. Madhyastha, T.M., Reed, D.A.: Input/output access pattern classification using hidden Markov models. In: Proceedings of the Fifth Workshop on Input/Output in Parallel and Distributed Systems, San Jose, CA, ACM Press (1997) 57–67
3. Lei, H., Duchamp, D.: An analytical approach to file prefetching. In: 1997 USENIX Annual Technical Conference, Anaheim, California, USA (1997)
4. Dodonov, E., Sousa, J.Q., Guardia, H.C.: Gridbox: securing hosts from malicious and greedy applications. In: Proceedings of the 2nd workshop on Middleware for grid computing, New York, NY, USA, ACM Press (2004) 17–22
5. Mello, R., Senger, L., Yang, L.: Automatic text classification using an artificial neural network. High Performance Computational Science and Engineering **1** (2005) 1–21
6. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Dagum, D., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Schreiber, R.S., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The nas parallel benchmarks. The International Journal of Supercomputer Applications **5**(3) (1991) 63–73
7. Cao, P., Felten, E.W., Karlin, A.R., Li, K.: A study of integrated prefetching and caching stategies. In: Proceedings of the 1995 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems, ACM Press (1995) 188–197
8. Dodonov, E., Guardia, H.C.: An architecture for integrated caching and prefetching mechanisms for distributed parallel file systems. In: Proceedings of the 2002 CLEI. (2002)
9. Reddy, A.L.N.: Evaluation of caching strategies for a multimedia storage server. In: International Conference on Multimedia Computing and Systems. (1997) 118–125
10. Cortes, T., Labarta, J.: Linear aggressive prefetching: A way to increase the performance of cooperative caches. In: Proceedings of the Joint International Parallel Processing Symposium and IEEE Symposium on Parallel and Distributed Processing, San Juan, Puerto Rico (1999) 45–54
11. Bianchini, R., Pinto, R., Amorim, C.L.: Data prefetching for software DSMs. In: International Conference on Supercomputing. (1998) 385–392
12. Mehrotra, S., Harrison, L.: Examination of a memory access classification scheme for pointer-intensive and numeric programs. In: ICS96. (1996) 133–140
13. Senger, L.J., Mello, R.F., Santana, M.J., Santana, R.C.: An on-line approach for classifying and extracting application behavior on linux. In Yang, L.T., Guo, M., eds.: High Performance Computing: Paradigm and Infrastructure. John Wiley and Sons (2005)
14. Carpenter, G.A., Grossberg, S.: Art 2: Selforganisation of stable category recognition codes for analog input patterns. Applied Optics **26** (1987) 4919–4930
15. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., Lang, K.: Phoneme recognition using time delay neural networks. IEEE Transactions on Accoustics, Speech and Signal Processing **37** (1989) 328–339
16. Burns, G., Daoud, R., Vaigl, J.: LAM: An Open Cluster Environment for MPI. In: Proceedings of Supercomputing Symposium. (1994) 379–386
17. Dodonov, E., de Mello, R.F., Yang, L.T.: A network evaluation for lan, man and wan grid environments. In Yang, L.T., Amamiya, M., Liu, Z., Guo, M., Rammig, F.J., eds.: EUC. Volume 3824 of Lecture Notes in Computer Science., Springer (2005) 1133–1146
18. Zell, A., Mache, N., Sommer, T., Korb, T.: Design of the snns neural network simulator. In Kaindl, H., ed.: 7. Österreichische Artificial-Intelligence-Tagung. Springer, Berlin, Heidelberg (1991) 93–102

Process Scheduling Using Ant Colony Optimization Techniques*

Bruno Rodrigues Nery[1], Rodrigo Fernandes de Mello[1],
André Carlos Ponce de Leon Ferreira de Carvalho[1], and Laurence Tianruo Yang[2]

[1] Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação – São Carlos, SP, Brazil
[2] St. Francis Xavier University, Antigonish, NS, Canada
lyang@stfx.ca

Abstract. The growing availability of low cost microprocessors and the evolution of computing networks have enabled the construction of sophisticated distributed systems. The computing capacity of these systems motivated the adoption of clusters to build high performance solutions. The improvement of the process scheduling over clusters originated several proposals of scheduling and load balancing algorithms. These proposals have motivated this work, which defines, evaluates and implements a new load balancing algorithm for heterogeneous capacity clusters. This algorithm, named Ant Scheduler, uses concepts of ant colonies for the development of optimization solutions. Experimental results obtained in the comparison of Ant Scheduler with other approaches investigated in the literature show its ability to minimize process mean response times, improving the performance.

1 Introduction

The growing availability of low cost microprocessors and the evolution of computing networks have enabled the construction of sophisticated distributed systems. In such systems, the processes executed on network computers communicate to each other to perform a collaborative computing task. A load balancing algorithm is frequently adopted to distribute the processes among computers.

A load balancing algorithm is responsible to equally distribute the processes load among the computers of an distributed environment [1]. Krueger and Livny [2] demonstrate that these algorithms can reduce the mean and standard deviation of processes' response times. Shorter response times result in higher performance in the execution of the processes.

The load balancing algorithms involve four policies: transference, selection, location and information [1]. The transference policy determines whether a computer is in a suitable state to participate in a task transfer, either as a server or as a receiver of processes. The selection policy defines the processes that should be transferred from the busiest computer to the idlest one. The location policy is responsible to find a suitable transfer partner (sender or receiver) for a computer, once the transfer policy has decided about

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 304–316, 2006.
© Springer-Verlag Berlin Heidelberg 2006

its state. A serving computer distributes the processes, when it is overloaded; a receiving computer requests processes, when it is idle. The information policy defines when and how the information regarding the computers' availability is updated in the system. Several works related to load balancing can be found in the literature [3,4,1,5,6].

Zhou and Ferrari [3] evaluated five server-initiated load balancing algorithms, i.e. initiated by the most overloaded computer: Disted, Global, Central, Random and Lowest. In Disted, when a computer suffers any modification in its load, it emits messages to the other computers to inform its current load. In Global, one computer centralizes all the computers' load information of the environment and sends broadcast messages in order to keep the other computers updated. In Central, as in Global, a central computer receives all the load information related to the system; however, it does not update the other computers with this information. This central computer decides the resources allocation in the environment. In Random, no information about the environment load is handled. Now, a computer is selected by random in order to receive a process to be initiated. In Lowest, the load information is sent when demanded. When a computer starts a process, it requests information and analyzes the loads of a small set of computers and submit the processes to the idlest one, the computer with the shortest process queue.

Theimer and Lantz [4] implemented algorithms similar to Central, Disted and Lowest. They analyzed these algorithms for systems composed of a larger number of computers (about 70). For the Disted and Lowest algorithms, a few process receiver and sender groups were created. The communication within these groups was handled by using a multicast protocol, in order to minimize the message exchange among the computers. Computers with load lower than a inferior limit participate of the process receiver group, whilst those with load higher than a superior limit participate of the process sender group.

Theimer and Lantz recommend decentralized algorithms, such as Lowest and Disted, as they do not generate single points of failure, as Central does. Central presents the highest performance for small and medium size networks, but its performance declines in larger environments. They concluded that algorithms like Lowest work with the probability of a computer being idle [4]. They assume system homogeneity, as they use the size of the CPU's waiting queue as the load index. The process behavior is not analyzed; therefore, the actual load of each computer is not measured.

Shivaratri, Krueger and Singhal [1] analyzed the server-initiated, receiver-initiated, symmetrically initiated, adaptive symmetrically initiated and stable symmetrically initiated algorithms. In their studies, the length of the process waiting queue at the CPU was considered as the load index. This measure was chosen because it is simple and, therefore, can be obtained with fewer resources. They concluded that the receiver-initiated algorithms present a higher performance than the server-initiated ones. In their conclusions, the algorithm with the highest final performance was the stable symmetrically initiated. This algorithm preserves the history of the load information exchanged in the system and takes actions to transfer the processes by using this information.

Mello *et al.* [5] proposed a load balancing algorithm for distributing processes on heterogeneous capacity computers. This algorithm, named TLBA (Tree Load Balancing

Algorithm), organizes computers in a virtual tree topology and starts delivering processes from the root to the leaves. In their experiments, this algorithm presented a very good performance, with low mean response time.

Senger *et al.* [6] proposed *GAS*, a genetic scheduling algorithm which uses information regarding the capacity of the processing elements, applications' communication and processing load, in order to allocate resources on heterogeneous and distributed environments. *GAS* uses Genetic Algorithms to find out the most appropriate computing resource subset to support applications.

These works, together with the development of new computing techniques based on biology, motivated the proposal of a new load balancing algorithm, named Ant Scheduler, which aims to apply Ant Colony Optimization techniques [7] to schedule processes on heterogeneous capacity computers. Experiments were carried out to evaluate the proposed algorithm and compare it with other algorithms found in the literature. The results confirm its applicability in heterogeneous cluster computing environments and suggest its potential as an alternative approach for load balancing.

This paper is divided into the following sections: section 2 briefly introduces the basic concepts of Ant Colony Optimization; section 3 describes the proposed load balancing algorithm based on Ant Colonies, named Ant Scheduler; The simulations performed and the results obtained are presented in section 4; section 5 describes the implementation of the proposed algorithm; Finally, section 6 has the main conclusions of this work.

2 Ant Colony Optimization

In the last years, there was a large growth in the research of computational techniques inspired in nature. This area, named Bio-inspired Computing, has provided biologically motivated solutions for several real world problems. Among the Bio-inspired Computing techniques, Artificial Neural Networks (ANN), Evolutionary Algorithms (EA), Artificial Immune Systems (AIS) and Ant Colony Optimization (ACO) can be mentioned.

One of the most promising of these techniques is ACO [7], a meta-heuristic technique based on the structure and behavior of ant colonies that has been successfully applied to several optimization problems [8,9].

Apparently simple organisms, ants can deal with complex tasks by acting collectively. This collective behavior is supported by the release of a chemical substance, named pheromone. During their movement, ants deposit pheromone in their followed paths. The presence of pheromone in a path, on its turn, attracts other ants. Thus, pheromone plays a key role in the information exchange between ants, allowing the accomplishment of several important tasks. A classical example is the selection of the shortest path between their nest and a food source.

In order to formally define ACO, assume four ants and two possible paths, P_1 and P_2 , which link a nest N_E to a food source F_S, such that $P_1 > P_2$. Initially, all the ants (A_1, A_2, A_3 and A_4) are in N_E and must choose between the paths P_1 and P_2 to arrive to F_S.

1. In N_E, the ants (A_1, A_2, A_3 and A_4) do not know the localization of the food source (F_S). Thus, they randomly choose between P_1 and P_2, with the same probability. Assume that ants A_1 and A_2 choose P_1, and ants A_3 and A_4 choose P_2.
2. While the ants pass by P_1 and P_2, they leave a certain amount of pheromone on the paths, τ_{C1} and τ_{C2}, respectively.
3. As $P_2 < P_1$, A_3 and A_4 arrive to F_S before A_1 and A_2. In this moment, $\tau_{C2} = 2$. Since A_1 and A_2 have still not arrived to F_S, $\tau_{C1} = 0$. In order to come back to N_E, A_3 and A_4 must choose again between P_1 and P_2. As in F_S, $\tau_{C2} > \tau_{C1}$, the probability of these ants choosing P_2 is higher. Assume that A_3 and A_4 choose P_2.
4. When A_3 and A_4 arrive to N_E again, τ_{C2} arrives to 4. This increase in τ_{C2} consolidates the rank of P_2 as the shortest path. When A_1 and A_2 arrive to F_S, $\tau_{C2} = 4$ and $\tau_{C1} = 2$. Thus, the probability of A_1 and A_2 coming back to N_E through P_2 becomes higher.

In the previous example, at the beginning, when there is no pheromone, an ant looking for food randomly chooses between P_1 and P_2 with a probability of 0.5 (50% of possibility for each path). When there is pheromone on at least one of the paths, the probability of selecting a given path is proportional to the amount of pheromone on it. Thus, paths with a higher concentration of pheromone have a higher chance of being selected.

It must be observed that most ACO approaches, in spite of being inspired by the problem solving paradigms found in biological ants, do not build replicas of them. Features of real ants may be absent and other additional techniques may be used to complement the use of pheromone.

One of the problems that may arise with the use of pheromone is the stagnation. Suppose, for example, that ants get addicted to a particular path. Sometimes in the future, that path may become congested, becoming nonoptimal. Another problem arises when a favorite path is obstructed and can no longer be used by the ants. In order to reduce this problem, the following approaches have been employed:

Evaporation: Reduce the pheromone values τ_i by a ρ factor to prevent high pheromone concentration in optimal paths, which avoids the exploration of other (new or better) alternatives.

Heuristic: This approach combines pheromone concentration τ_i and a heuristic function η_i. The relative importance of each information is defined by two parameters, α and β.

3 Ant Scheduler

The problem of process allocation in heterogeneous multi-computing environments can be modeled by using graphs. In this case, each process request for execution has the nodes S and T as origin and destination, respectively. S and T are connected by N different paths, each corresponding to a computer in a cluster. This graph is employed to improve the general performance of the system by minimizing the mean congestion of the paths.

The good results obtained by ACO in graph-based problems favor the use of ACO for the optimization of process allocation on heterogeneous cluster computing environments. For such, each initiated process can be seen as an ant looking for the best path starting in the nest in order to arrive as fast as possible to the food source. In this search, each computer can be seen as a path and the conclusion of the program execution as the food source.

The Ant Scheduler algorithm is based on the *ant-cycle* proposed by Dorigo *et al.* [7]. When the computer responsible for the distribution of processes (master) in the cluster is started, each edge in the graph has its pheromone intensity initiated with a value $\tau_i = c$. When a process is launched, it is seen as an ant able to migrate. Thus, this process must select one of the paths (the computers of the cluster) to its destination (complete execution). The probability of an ant choosing a path i is given by equation 1, where τ_i is the pheromone level on path i, η_i is a value associated to the computer i by a heuristic function, and the parameters α and β control the relevance of τ_i and η_i:

$$p_i = \frac{\tau_i^{\alpha} \cdot \eta_i^{\beta}}{\sum_{j=1}^{N} \tau_j^{\alpha} \cdot \eta_j^{\beta}}, \tag{1}$$

$$\Delta_i = \Delta_i + \frac{Q}{T}, \tag{2}$$

$$\tau_i(t+1) = \rho \cdot \tau_i(t) + \Delta_i. \tag{3}$$

In this paper, this heuristic function is proportional to the load of the ith computer. The denominator is the sum of the pheromone levels weighted by the heuristic function and controlled by the parameters α and β. When an ant arrives to its destination (when a process finishes), it deposits a Δ amount of pheromone in the covered path (equation 2: where Q is a constant and T is the time spent by the ant to arrive at its destination (the process running time)).

In order to prevent an addiction to a particular computer, the paths face continuous pheromone evaporation. Thus, in regular time intervals, the amount of pheromone changes according to the rule of equation 3, where ρ is a coefficient such that $(1 - \rho)$ represents the pheromone evaporation between t and $t + 1$. Additionally, Δ_i is reseted ($\Delta_i = 0$) in regular time intervals.

One problem with this approach is the possibility of a poor performance due to the different range of values for τ_i and η_i. In order to overcome this problem, these values are normalized using a logarithmic scale, modifying the equation 1 and originating the equation 4:

$$p_i = \frac{(\log \tau_i)^{\alpha} \cdot (\log \eta_i)^{\beta}}{\sum_{j=1}^{N} (\log \tau_j)^{\alpha} \cdot (\log \eta_j)^{\beta}}. \tag{4}$$

Another problem found was the frequent allocation of a low value, between 0 and 1, to τ_i, making $\log \tau_i < 0$, leading to unrealistic values for the probability function. This problem was solved by using $\log \epsilon + \tau_i$ instead of $\log \tau_i$, where $\epsilon = 1$. This resulted to the equation 5:

$$p_i = \frac{(\log \epsilon + \tau_i)^\alpha \cdot (\log \epsilon + \eta_i)^\beta}{\sum_{j=1}^{N} (\log \epsilon + \tau_j)^\alpha \cdot (\epsilon + \log \eta_j)^\beta}. \quad (5)$$

Algorithm 1. Ant Scheduler: process started

Choose a computer with probability p_i, calculated using equation 5
Schedule process on chosen computer

Algorithm 2. Ant Scheduler: process finished

Update the amount of pheromone Δ_i using equation 2

Algorithm 3. Ant Scheduler: pheromone evaporation

loop
  for all i such that i is a cluster node **do**
    Update the amount of pheromone τ_i using equation 3
    Reset the amount of pheromone Δ_i ($\Delta_i = 0$)
  end for
end loop

The Ant Scheduler is composed of the Algorithms 1, 2 and 3. The first algorithm is executed when a new process, with possibility of migration, is initiated. When a process completes its execution, the second algorithm starts. The third algorithm is periodically executed, in order to perform the pheromone evaporation.

4 Simulation Results

Several experiments have been carried out on environments with 32 computers for the evaluation of the Ant Scheduler algorithm behavior. The Ant Scheduler parameters used were $\alpha = 1$, $\beta = 1$, $\rho = 0.8$ and $Q = 0.1$. Parallel applications of up to 8, 64 and 128 tasks have been evaluated. This configuration allows the evaluation of the algorithm in situations where there are many tasks synchronized with others, that is, tasks that communicate among themselves to solve the same computing problem.

The workload imposed by such applications follows the workload model by Feitelson[1][10]. This model is based on the analysis of six execution traces of the following production environments: 128-node iPSC/860 at NASA Ames; 128-node IBM SP1 at

[1] http://www.cs.huji.ac.il/labs/parallel/workload/models.html

Argonne; 400-node Paragon at SDSC; 126-node Butterfly at LLNL; 512-node IBM SP2 at CTC; 96-node Paragon at ETH.

According to this model, the arrival of processes is derived from an exponential probability distribution function (pdf) with mean equal to $1,500$ seconds. This model was adopted to simulate and allow a comparative evaluation of Ant Scheduler and other algorithms found in the literature.

In order to carry out the experiments and evaluate the scheduling algorithm proposed in this study, the authors used the model for creation of heterogeneous distributed environments and evaluation of the parallel applications response time - UniMPP (*Unified Modeling for Predicting Performance*) [11]. The adopted model is able to generate the mean execution time of the processes submitted to the system. The mean response time is generated after reaching the confidence interval of 95%.

In this model, every processing element (PE), PEM_i, is composed of the sextuple $\{pc_i, mm_i, vm_i, dr_i, dw_i, lo_i\}$, where pc_i is the total computing capacity of each computer measured in instructions per time unit, mm_i is the main memory total capacity, vm_i is the virtual memory total capacity, dr_i is the hard disk reading throughput, dw_i is the hard disk writing throughput, and lo_i is the time between sending and receiving a message.

In this model, every process is represented by the sextuple $\{mp_j, sm_j, pdfdm_j, pdfdr_j, pdfdw_j, pdfnet_j\}$, where mp_j represents the processing consumption, sm_j is the amount of static memory allocated by the process, $pdfdm_j$ is the probability distribution for the memory dynamic occupation, $pdfdr_j$ is the probability distribution for file reading, $pdfdw_j$ is the probability distribution for file writing, and $pdfnet_j$ is the probability distribution for messages sending and receiving.

In order to evaluate the Ant Scheduler algorithm, a class was included in the object-oriented simulator [2] [11]. This class implements the functionalities of Ant Scheduler and has been aggregated to the UniMPP model simulator to generate the mean response times of an application execution. These results were used to evaluate the performance of Ant Scheduler and to allow comparisons with other algorithms.

4.1 Environment Parameterizations

Experiments were conduced in environments composed of 32 computers. In these experiments, each PEM_i for the UniMPP model was probabilistically defined. The parameters pc_i, mm_i, vm_i, dr_i, dw_i were set by using an uniform probability distribution function with the mean of $1,500$ Mips (millions of instructions per second), $1,024$ MBytes (main memory), $1,024$ MBytes (virtual memory), 40 MBytes (file reading transference rate from hard disk) and 30 MBytes (file writing transference rate to hard disk). These measures were based on the actual values obtained using a group of machines from our research laboratory (Distributed Systems and Concurrent Programming Laboratory). These measures followed the benchmark proposed by Mello and Senger [11][3]. These parameter values and the use of probability distributions allow the creation of heterogeneous environments to evaluate the Ant Scheduler algorithm.

[2] SchedSim - available at website http://www.icmc.usp.br/~mello/outr.html

[3] Available at http://www.icmc.usp.br/~mello/

The Feitelson's workload model was used to define the occupation parameter (in Mips) of the processes (or tasks) that take part of the parallel application. The remaining parameters required for the UniMPP to represent a process were defined as: sm_j, the amount of static memory used by the process, based on an exponential distribution with a mean of 300 KBytes; $pdfdm_j$, the amount of memory dynamically allocated, defined by an exponential distribution with a mean of $1,000$ KBytes; $pdfdr_j$, the file reading probability, defined by an exponential distribution with a mean of one read at each $1,000$ clock ticks, same value used to parameterize the writing in files ($pdfdw_j$); $pdfnet_j$, the receiving and sending of network messages, parameterized by an exponential distribution with a mean of one message at each $1,000$ clock ticks.

During the experiments, all computers were located at the same network, as a ordinary cluster. Within the network, the computers present a delay (RTT - Round-Trip Time according to the model by Hockney [12]) of 0.0001 (mean value extracted by the *net* benchmark by Mello and Senger [11] for a Gigabit Ethernet network).

4.2 Algorithms Simulated

The performance of Ant Scheduler is compared with 5 other scheduling and load balancing algorithms proposed in literature: `DPWP` [13], `Random`, `Central`, `Lowest` [3], `TLBA` [5] and `GAS` [6].

The `DPWP` (*Dynamic Policy Without Preemption*) algorithm performs the parallel applications scheduling taking into account a distributed and heterogeneous execution scenario [13]. This algorithm allows the scheduling of the applications developed on PVM, MPI and CORBA. The details involved in the task attributions are supervised by the scheduling software, AMIGO [14][4].

The load index used in this algorithm is the queue size of each PE (processing element). Through this index, the load of a PE is based on the ratio between its number of tasks being executed and its processing capacity. The processing capacity is measured by specific *benchmarks* [14,15]. The `DPWP` scheduling algorithm uses load indexes to create an ordered list of PEs. The parallel application tasks are attributed to the PEs of this list, in a circular structure.

The `Lowest`, `Central` and `Random` algorithms were investigated for load balancing in [3]. These algorithms are defined by two main components: LIM (*Load information manager*) and LBM (*Load balance manager*). The first component is responsible for the information policy and for monitoring the computers' load in order to calculate the load indexes. The latter defines how to use the collected information to find out the most appropriate computer to schedule processes. The approach followed by these components to perform their tasks allows the definition of distinct algorithms. These algorithms differ from the scheduling algorithms by being designed to perform the load balance, thus there is no global scheduling software to which the applications are submitted. In fact, each PE should locally manage the application tasks that reach it, initiating them locally or defining how another PE will be selected to execute tasks.

[4] We have compared our results to this work, because it was also developed in our Laboratory.

The Lowest algorithm aims to achieve the load balance by minimizing the number of messages exchanged among its components. When a task is submitted to the environment, the LIM receiving the request defines a limited set of remote LIMs. The loads of the PEs of this set are received and the idlest PE is selected to receive the task.

The Central algorithm employs a master LBM and a master LIM. Both of them centralize the decision making related to the load balance. The master LIM receives the load indexes sent by the slave LIMs. The master LBM receives the requests to allocate processes to the system and uses the information provided by the master LIM to make these allocations.

The Random algorithm does not use information regarding the system load to make decisions. When a task is submitted to the execution environment, the algorithm randomly selects an PE. The load index used by the Lowest and Central algorithms is calculated based on the number of processes in the execution queue. Zhou and Ferrari [3] have observed that the Lowest and Central algorithms present similar performance and that the Random algorithms present the worst results of all. They also suggested the Lowest algorithm for distributed scenarios, because it is not centralized.

The TLBA (*Tree Load Balancing Algorithm*) algorithm aims at balancing loads in scalable heterogeneous distributed systems [5]. This algorithm creates a logical interconnection topology with all PEs, in a tree format, and performs the migration of tasks in order to improve the system load balance.

The GAS (*Genetic Algorithm for Scheduling*) algorithm uses Genetic Algorithms to propose optimized scheduling solutions [6]. The algorithm considers knowledge about the execution time and applications' behavior to define the most adequate set of computing resources to support a parallel application on a distributed environment composed of heterogeneous capacity computers. GAS uses the crossover and mutation operators to optimize the probabilistic search for the best solution for a problem. Based on Genetics and Evolution, Genetic Algorithms are very suitable for global search and can be efficiently implemented in parallel machines.

4.3 Comparison with Other Algorithms

For the validation of the Ant Scheduler algorithm, its performance was compared with results obtained by the five algorithms previously described. For such, the authors carried out simulations where all these algorithms were evaluated running parallel applications composed of different number of tasks. Figures 1.a and 1.b show the process mean response times for parallel applications with up to 64 and 128 tasks, respectively.

Random had the worst results, while Ant Scheduler presented the best performance. The poor performance obtained by GAS can be explained by the fact that its execution time increases according to the number of computers. This occurs due to the use of larger chromosomes (this approach is based on Genetic Algorithms), which have to be evaluated by the fitness function. This evaluation requires a long time, which is added to the scheduling cost, jeopardizing the process execution time. It is hard to observe the curve for Ant Scheduler in Figure 1.a due to the small mean response times in comparison with the other algorithms.

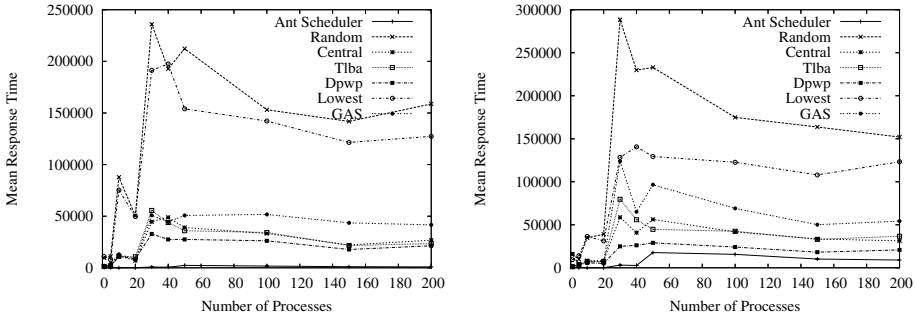


Fig. 1. Simulation results

These results show that, in all the scenarios investigated, the Ant Scheduler presented the best performance.

5 Implementation

In order to allow real experiments, Ant Scheduler was implemented using the Linux kernel 2.4.24. This implementations uses the process migration service of the openMosix[5] patch. openMosix is a software designed to balance the load of clusters by distributing processes.

The implementation was performed by adding a set of traps inside the Linux kernel. The first trap was implemented in the system call *do_fork*. Whenever a new process is started, *do_fork* is called. This system call executes the first trap of Ant Scheduler, which chooses the node where the new process will run. This phase is based on the pheromone level and the computing capacity of each node. Similarly, when a process finishes, the system call *do_exit* is made. This system call executes the second trap of Ant Scheduler, which updates the amount of pheromone in the computer (ant's path) where the process was running.

These traps were implemented in a kernel module by using function pointers, allowing simple changes to use another process scheduling algorithm. When the module is loaded, it registers its functions (traps). This module also starts a thread that periodically updates the pheromone level of each computer applying the equation 3.

Experiments were carried out to evaluate the process execution time for an environment using Ant Scheduler and openMosix on a set of five Dual Xeon 2.4 Ghz computers. Table 1 presents the results in process mean execution time (in seconds) for a load of 10 low-load, 10 mean-load and 10 high-load applications executing simultaneously. According to these results, the use of Ant Scheduler reduced the mean response time.

[5] Openmosix is a Linux kernel patch developed by Moshe Bar which allows automatic process migration in a cluster environment – Available at http://openmosix.sourceforge.net/

Table 1. Experimental Results

Experiment	without Ant Scheduler	with Ant Scheduler
1	351.00	327.00
2	351.00	336.00
3	351.00	318.00
4	354.00	321.00
5	351.00	318.00
6	351.00	333.00
7	351.00	321.00
8	351.00	336.00
9	348.00	309.00
10	348.00	315.00
Mean	350.70	323.40
Std Dev	1.615	8.777

In order to evaluate the statistical significance of the results obtained, the authors applied the Student's t-test. In this analysis, the authors used the standard error s_x for small data samples [16], given by equation $s_x = \frac{s}{\sqrt{n}}$.s is the standard deviation and n is the number of samples. Applying the equation, the standard errors of 0.51 and 2.775 were obtained without Ant Scheduler and with Ant Scheduler, respectively.

In the test, the authors propose the null hypothesis (from hypothesis test) $H_0 : \mu_{with} = \mu_{without}$, with the alternative hypothesis $H_A : \mu_{with} < \mu_{without}$ to evaluate whether the results are statistically equivalent. The hypothesis H_0 considers the results of the Ant Scheduler and the standard openMosix to be similar. If the test is rejected, the alternative hypothesis H_A is accepted. This hypothesis considers the process mean response time for the environment adopted. The processes are distributed using the Ant Scheduler is lower, what confirms the superiority of Ant Scheduler.

The significance level used for one-tailed test is $\alpha = 0.0005$. μ_{with} is the process mean response time with Ant Scheduler; $\mu_{without}$ is the process mean response time with the standard openMosix. For the adopted significance level α, the data sets have to present a difference of at least 4.781 in the t-test to reject the hypothesis. This value is found in tables of critical values for the *t*-student distribution.

Applying the equation $t = \frac{\mu_{without} - \mu_{with}}{s_x}$, the value 9.83 is found, confirming that the results present statistic differences with $p < 0.005$, rejecting the hypothesis H_0. In this way the hypothesis H_A is valid and the system with Ant Scheduler presents better results than standard openMosix.

By applying statistic tools[6] over the data sets, it is possible to find the most precise $\alpha = 0.0000018$ for a one-tailed test. This value shows how many times the alternative hypothesis is true. In this case, H_A can be considered true in $9,999,982$ of $10,000,000$ executions, showing that Ant Scheduler reduces the response time. Only in 18 of these

[6] The authors applied the Gnumeric tool in this analysis – a free software tool licensed under GNU/GPL terms.

executions the results of Ant Scheduler and openMosix did not present significant statistical differences.

6 Conclusions

In this work, the authors proposed a new approach for load balance in heterogeneous computers using an algorithm based on Ant Colony Optimization. The proposed approach was motivated by the successes obtained by this technique in several real world problems.

This algorithm, named Ant Scheduler, had its performance compared with five other algorithms found in the literature. The simulation results, where the proposed algorithm presented a performance superior to the other algorithms investigated, suggest the potential of the Ant Scheduler algorithm for heterogeneous cluster computing environments. These results have motivated its implementation to validate the theoretical concepts. The algorithm was implemented in a real Linux environment. This implementation was evaluated through experiments and compared with the standard scheduling in openMosix. In these experiments, Ant Scheduler also has presented good results.

Acknowledgement

The authors thank to the Fapesp Brazilian Foundation (process number 2004/02411-9).

References

1. Shivaratri, N.G., Krueger, P., Singhal, M.: Load distributing for locally distributed systems. IEEE Computer **25**(12) (1992) 33–44
2. Krueger, P., Livny, M.: The diverse objectives of distributed scheduling policies. In: Seventh Int. Conf. Distributed Computing Systems, Los Alamitos, California, IEEE CS Press (1987) 242–249
3. Zhou, S., Ferrari, D.: An experimental study of load balancing performance. Technical Report UCB/CSD 87/336, PROGRES Report N.o 86.8, Computer Science Division (EECS), Universidade da California, Berkeley, California 94720 (1987)
4. Theimer, M.M., Lantz, K.A.: Finding idle machines in a workstation-based distributed system. IEEE Transactions on Software Engineering **15**(11) (1989) 1444–1458
5. Mello, R.F., Trevelin, L.C., Paiva, M.S., Yang, L.T.: Comparative analysis of the prototype and the simulator of a new load balancing algorithm for heterogeneous computing environments. In: International Journal of High Performance Computing and Networking. Volume 1, No.1/2/3. Interscience (2004) 64–74
6. Senger, L.J., de Mello, R.F., Santana, M.J., Santana, R.H.C., Yang, L.T.: Improving scheduling decisions by using knowledge about parallel applications resource usage. In: Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC-05), Sorrento, Naples, Italy (2005) 487–498
7. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics-Part B **26** (1996) 1,13

8. Shmygelska, A., Hoos, H.H.: An ant colony optimisation algorithm for the 2d and 3d hydrophobic polar protein folding problem. BMC Bioinformatics (2005) 1–22
9. Acan, A.: An external memory implementation in ant colony optimization. In: Proceedings of the 4th International Workshop on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium (2004) 73–82
10. Feitelson, D.G., Jette, M.A.: Improved utilization and responsiveness with gang scheduling. In Feitelson, D.G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Springer (1997) 238–261 Lect. Notes Comput. Sci. vol. 1291.
11. de Mello, R.F., Senger, L.J.: Model for simulation of heterogeneous high-performance computing environments. In: 7th International Conference on High Performance Computing in Computational Sciences – VECPAR 2006, Springer-Verlag (2006) 11
12. Hockney, R.W.: The Science of Computer Benchmarking. Soc for Industrial & Applied Math (1996)
13. Araújo, A.P.F., Santana, M.J., Santana, R.H.C., Souza, P.S.L.: DPWP: A new load balancing algorithm. In: ISAS'99, Orlando, U.S.A. (1999)
14. Souza, P.S.L.: AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos. PhD thesis, IFSC-USP (2000)
15. Santos, R.R.: Escalonamento de aplicações paralelas: Interface amigo-corba. Master's thesis, Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo (2001)
16. W.C.Shefler: Statistics: Concepts and Applications. The Benjamin/Cummings (1988)

Interlocking Control by Distributed Signal Boxes: Design and Verification with the SPIN Model Checker

Stylianos Basagiannis, Panagiotis Katsaros, and Andrew Pombortsis

Department of Informatics, Aristotle University of Thessaloniki,
54124 Thessaloniki, Greece
{basags, katsaros, apombo}@csd.auth.gr

Abstract. Control systems are required to comply with certain safety and liveness correctness properties. In most cases, such systems have an intrinsic degree of complexity and it is not easy to formally analyze them, due to the resulting large state space. Also, exhaustive simulation and testing can easily miss system errors, whether they are life-critical or not. In this work, we introduce an interlocking control approach that is based on the use of the so-called *Distributed Signal Boxes (DSBs)*. The proposed control design is applied to a railway-interlocking problem and more precisely, to the Athens underground metro system. Signal boxes correspond to the network's interlocking points and communicate only with their neighbor signal boxes. Communication takes place by the use of rendezvous communication channels. This design results in a simple interlocking control approach that compared to other centralized solutions produces a smaller and easier to analyze state space. Formal analysis and verification is performed with the SPIN model checker.

Keywords: interlocking control, safety, distributed control, model checking.

1 Introduction

Interlocking control aims to prevent certain operations from occurring, unless preceded by certain events. Although interlocking control may be used as a general signaling technique in the design of e.g. telecom network management systems, the term usually refers to a range of vehicular traffic control applications. Interlocking systems have been mainly developed and studied in the field of railway traffic control, where their task is to prevent trains from colliding and derailing, while at the same time allowing their movements.

Whether interlocking systems are integrated in life-critical control systems or not they are required to comply with certain safety and liveness correctness properties. This fact elevates formal modeling and model checking to the number one concern in the design and development of real-scale interlocking systems.

However, in most cases, these systems have an intrinsic degree of complexity and it is not easy to fully analyze them, due to the resulting large state space. Usually, the control logic of the interlocking is not the single design concern that has to be checked with respect to the required safety and liveness properties. Complete system designs have to include also an adequate representation of the communication between the system's components, as well as, the applied fault tolerance approach.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 317 – 328, 2006.
© Springer-Verlag Berlin Heidelberg 2006

This work introduces the control logic of a new interlocking approach that is based on the use of the so-called *Distributed Signal Boxes* (DSBs). The proposed interlocking control is applied to a real-scale system model and more precisely to the recently build Athens underground metro network. Signal boxes correspond to the network's interlocking points and communicate only with their neighbors. Communication takes place by the use of rendezvous communication channels. The proposed interlocking control invests on design simplicity and avoids proprietary concept definitions and proprietary system requirements. It is not difficult to be generalized in networks with arbitrary topologies and compared to other centralized solutions produces a smaller and easier to analyze state space, as well as, improved scalability prospects. Formal analysis and verification is performed with the SPIN model checker.

Section 2 surveys recent research in interlocking control and attempts a comparison with the proposed solution. Section 3 introduces our interlocking approach and the use of the so-called *Distributed Signal Boxes*. Section 4 refers to the formal verification of the proposed interlocking control. The paper ends with a discussion on our work's potential impact and comments interesting future research prospects.

2 Related Work

In related work, interlocking control is mainly studied in the context of railway signaling systems. The work published in [1] points out the lack of precise concept definitions and the lack of overall system requirements. The author proposes an approach to formalize the principles and the concepts of interlocking systems in VDM (Vienna Development Method). However, he focuses on the Danish interlocking systems and underlines that interlocking systems from other countries may be different.

In [2], the authors analyze the safety of a real computer interlocking system, for the control of railway stations. The system's architecture is based on redundancy and is composed of a central nucleus connected to peripheral posts for the control of physical devices. A formal model of the system's safety logic was developed in Verus ([3]), a tool that combines symbolic model checking and quantitative timing analysis. The model was checked with respect to a number of safety and liveness properties that were included in the initial system's specifications. The safety logic of the same system was also modeled in [4] and [5], where the authors used the SPIN model checker ([6]) to analyze all system's functions that may be requested by an external operator.

SPIN was also used in [7] where the authors present a model of the same system and validate certain safety properties, in the presence of Byzantine system components or of some hardware temporary faults.

In [8], the authors introduce a model for the interlocking of a particular track layout that is the one used by an Australian railway operator. Interlocking control is coded in the so-called control tables and the described analysis aims to find erroneous or incomplete entries in the used tables. Modeling and safety checking is performed with the NuSMV model checker, but in earlier works the same group used also a Communicating Sequential Processes (CSP) approach and the Failure Divergence Refinement (FDR) model checker.

The work published in [9] reports the safety checking of the Line Block interlocking system that also adopts a centralized design approach. The overall control strategy runs on a Central Control Unit that communicates with a number of Peripheral

Control Units (PCUs). PCUs are expected to drive particular interlocking system components and detect external events.

There is only one attempt known to us, for the development and verification of a distributed interlocking system. In that work ([10]), the authors note that today's centralized interlocking systems are far too expensive for small or possibly private networks. They propose to distribute the tasks of train control, train protection and interlocking over a network of cooperating components, using the standard communication facilities offered by mobile telephone providers. Their approach is based on the use of the so-called switch boxes, which locally control the point where they are allocated. Train engines are carriers of train control computers, which collect the local state information from switch boxes along the track to derive the decision whether the train may enter the next track segment.

In contrast to the forenamed solution, our approach is based on signal exchanges between the Distributed Signal Boxes (DSBs). There is no need of a mobile communication medium, which in any case requires security and reliability mechanisms that are unnecessary for systems transmitting signals over wires. DSBs communicate only with their neighbors. This design principle results in a general peer-to-peer signaling approach, possible to be applied in a wide range of interlocking problems, other than the typical railway traffic control applications (see for example [11]).

3 Distributed Signal Boxes

DSBs are allocated to the network's interlocking points. Interlocking points communicate only with their corresponding DSB and DSBs communicate only with their neighbor DSBs. Communication takes place by the use of rendezvous communication channels. The proposed interlocking control has been successfully checked in network topologies that include two different types of communication links and more precisely the ones shown in Figure 1.

Fig. 1. DSBs communication links

In a given network that includes one or both types of communication links, interlocking control requires exchange of two distinct messages, which in fact are used as control flags of the network's traffic. These messages control the network resources and distribute them among the entities that request them. For the track layout of the recently built Athens underground metro (Figure 2) we also call these two messages *signals*. Signals control movement and track allocation by exchanging their locations in a series of consecutive DSBs communications. Interlocking points (stations) cannot communicate with their neighbors by direct communication links, since this is possible only through their corresponding signal boxes.

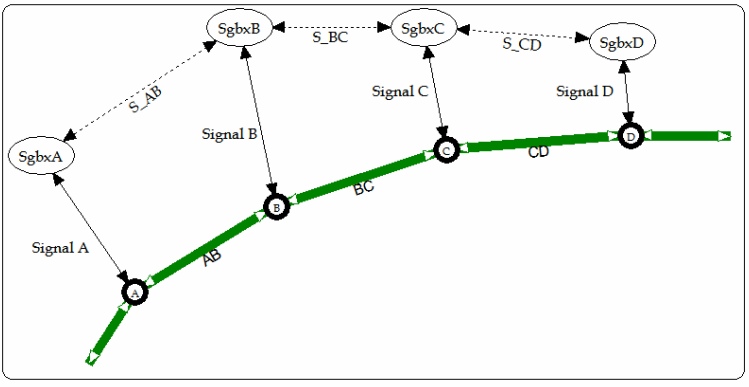


Fig. 2. DSB communication for the track layout of the Athens underground metro system

The system's network topology is progressively formed through a step-by-step introduction of new interlocking points (stations), in a ring or a tree-based structure (refer to Figure 1). For every station, we assign the tracks that connect it with its neighbors, from now on called *tunnels*, as well as, the communication channel (e.g. `Signal A`) used to exchange messages with the corresponding signal box. The topology of the modeled network depends on a mutually consistent declaration of the neighboring stations, for all stations that are included in the network.

The proposed interlocking control mechanism (Figure 3) is coded in only three procedures. One of them refers to the control logic of the network's stations. Another one specifies the control logic of the network's signal boxes and the last one inserts a request for one of the network's tunnels. The last mentioned procedure is used to initialize the model with the required number of trains.

The network perceives an initial train entrance to one of its tunnels by having required the relevant procedure to not proceed to its execution, up to the reception of the expected `msg2` signal. This control signal will be generated by the tunnel's entrance DSB process and as a result it will release the train to enter to the tunnel. The same train will be set again to a stop-wait state as a result of control signals exchanged between the network's DSBs and the station, where the train has arrived. In Figure 3 we used the SPIN's PROMELA language syntax to show a graphical representation of the described control signal exchanges.

4 Model Building and Model Checking Correctness Properties

In this section, we present a model building and model checking approach for the described interlocking control mechanism. We aim to prove that the proposed DSBs based solution meets certain safety and liveness properties, which are a requisite for its deployment to real-scale interlocking problems. We decided to use the SPIN model checker ([12], [13]) for the following reasons:

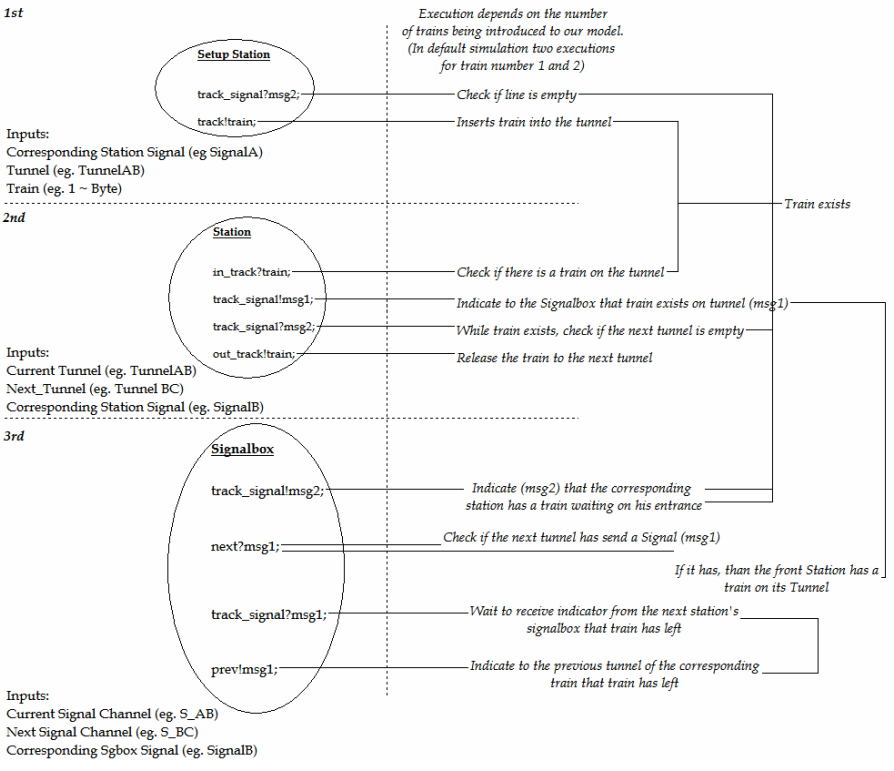


Fig. 3. The DSBs based interlocking mechanism

- SPIN has been successfully used in simulating, verifying and finding errors (counter examples) in a wide range of concurrent software systems.
- SPIN is a versatile model checker that provides support to report all detected deadlocks and livelocks, potential race conditions, as well as, possibly unwanted situations regarding the relative speeds of the concurrent processes. It makes possible to express and model check the expected safety and liveness properties as Liner Temporal Logic (LTL) formulae.
- SPIN model specification is expressed in PROMELA, a high-level specification language that provides built-in support for rendezvous, as well as, buffered message passing between the modeled processes.

4.1 Model Structure and Implementation

In order to apply the developed mechanism to the Athens underground metro, we first have to examine the topology of the railway network. There are three lines that intersect each other in four different stations (Figure 4). Each train moves within a certain line, but lines operate in different levels and do not interfere with other lines. We focus on modeling the station topology of Line 1 as a set of bi-directional interconnected tunnels. The used tunnels are shared by the trains moving in this line independently of the train movements in Lines 2 and 3.

Figure 4 shows the DSBs placement and the derived signal box connectivity for the entire Athens underground metro network. We assume installation of separate interlocking points within terminal stations, where the trains change direction. These additional interlocking points are called *line switching points* and come together with their corresponding signal box. Line 1 model structure is based on the signal box topology of the stations shown in Line 1. DSBs communicate with their corresponding stations through channels of rendezvous communication that are represented as channels of size zero, in order to not store any signal (`chan Signalx` declarations of Figure 5). When a train arrives to a station this station's DSB communicates with its neighbor DSBs by synchronized channels of communication (`chan S_xy` declarations of Figure 5), in order to check the availability of the outgoing tunnel. As we already noted, Line 1 (and all other lines) consists of a set of bi-directional tunnels where each direction route is represented by a separate communication channel of size 2 (`chan Tunnelxy` and `chan Tunnelyx` declarations of Figure 5).

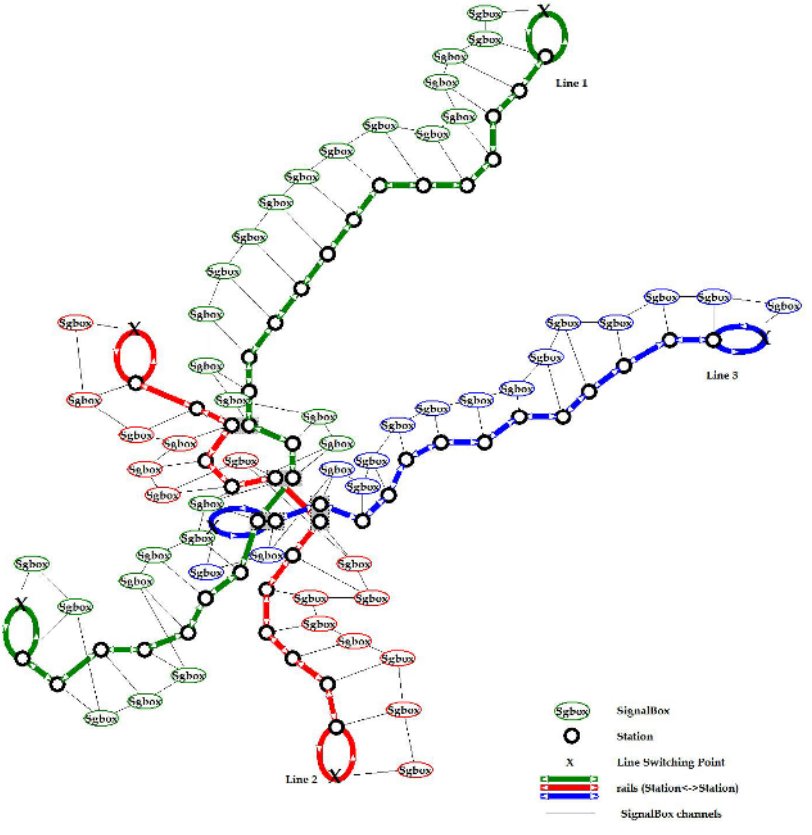


Fig. 4. The DSBs based interlocking mechanism for the Athens underground metro network

Tunnels and signal boxes interconnection is specified by the `run Setup`, the `run Station` and the `run Signalbox` procedure calls shown in Figure 7. The code of the forenamed procedures is given in Figure 6.

For a given station, say X, a train arrival to it (`proctype Station`) causes the dispatch of `msg1` to its signal box. X's signal box forwards `msg1` to the signal box of the previous station (`proctype Signalbox`). Regarding X's interlocking control, on reception of `msg2` the train enters into the next tunnel (`proctype Station`). However, `msg2` cannot be received if the signal box is still blocked, waiting for the dispatch of `msg1` from the next station's signal box (`proctype Signalbox`). Accordingly, this depends on the availability of X's outgoing tunnel, that is, it is possible only if another train has already left the next station or only upon the departure of that train from the next station (`proctype Station`).

```
/* Channels between Stations (Tunnels) */

chan TunnelAB=[2] of {byte};
chan TunnelBC=[2] of {byte};
chan TunnelCD=[2] of {byte};
chan TunnelDE=[2] of {byte};
.............................
chan TunnelXSw1=[2] of {byte};
.............................
chan TunnelBA=[2] of {byte};

/* Channels between Stations and Signalboxes */

chan SignalA=[0] of {mtype};
chan SignalB=[0] of {mtype};
chan SignalC=[0] of {mtype};
chan SignalD=[0] of {mtype};
.............................
chan SignalX=[0] of {mtype};

/* Channels between the Signalboxes */

chan S_AB=[0] of {mtype};
chan S_BC=[0] of {mtype};
chan S_CD=[0] of {mtype};
chan S_DE=[0] of {mtype};
.........................
chan S_XSw1=[0] of {mtype};
.........................
chan S_BA=[0] of {mtype};
```

Fig. 5. PROMELA declarations of tunnels, DSB to station and DSB to DSB channels

```
proctype Setup(chan track,track_signal;byte train)
{
track_signal?msg2;
track!train;
}

proctype Station(chan in_track,out_track,track_signal)
{
byte train;
do
 :: in_track?train;track_signal!msg1;
    track_signal?msg2;out_track!train;
od
}

proctype Signalbox(chan prev,next,track_signal)
{
do
 :: track_signal!msg2;next?msg1;
 :: track_signal?msg1;prev!msg1;
od
}
```

Fig. 6. DSBs interlocking control procedures

Figure 7 shows the PROMELA code for a model instance with 2 trains (two `Setup` procedure calls). It is not difficult to verify the expected safety and liveness properties for an arbitrary number of trains. A detailed operational view of the proposed interlocking control for a Line 1 segment is given in Figure 8.

```
/* Introduces train 1 into TunnelBC */

run Setup(TunnelBC, SignalB,1);

/* Introduces train 2 into TunnelML */

run Setup(TunnelML, SignalM,2);

/* assertion of the problem */

run Monitor();

/* StationA with track_signal A etc*/

run Station(TunnelSwA,TunnelAB, SignalA);
run Station(TunnelAB,TunnelBC, SignalB);
run Station(TunnelBC,TunnelCD, SignalC);
run Station(TunnelCD,TunnelDE, SignalD);
.....................................
run Station(TunnelNM,TunnelML,SignalM);
run Station(TunnelML,TunnelLK,SignalL);
.....................................

/*Signalbox A, communication channels, track_signal A etc*/

run Signalbox(S_SwA,S_AB,SignalA);
run Signalbox(S_AB,S_BC,SignalB);
run Signalbox(S_BC,S_CD,SignalC);
run Signalbox(S_CD,S_DE,SignalD);
.................................
run Signalbox(S_NM,S_ML,SignalM);
run Signalbox(S_ML,S_LK,SignalL);
.................................
```

Fig. 7. Initiating the DSBs interlocking control model

4.2 Model Checking Safety and Liveness Correctness Properties

The basic safety correctness property refers to the possibility of collision between the two trains in one of the tunnels' routes shown in Figure 8. The developed model makes this possible by having declared all tunnels' routes as separate communication channels of size 2 (Figure 5). The proposed interlocking control aims to prevent the two trains from occupying both channel positions at the same time. Correctness with respect to the forenamed safety property is checked by the following assertion:

ASR1: "*A tunnel route can only be occupied by one train at a time*"

Assertion *ASR1* is included in the code of procedure Monitor (Figure 9). Model checking is activated by the `run Monitor()` procedure call of Figure 7. In the performed full state space search of Figure 10 an assertion violation would be reported as error, but the obtained results prove the safety correctness of the proposed interlocking control (errors: 0). The reported number of states refers to a real-scale

application of our solution to a network of 26 interlocking points, 2 directions and 2 trains moving in both directions of the network's interconnected tunnels.

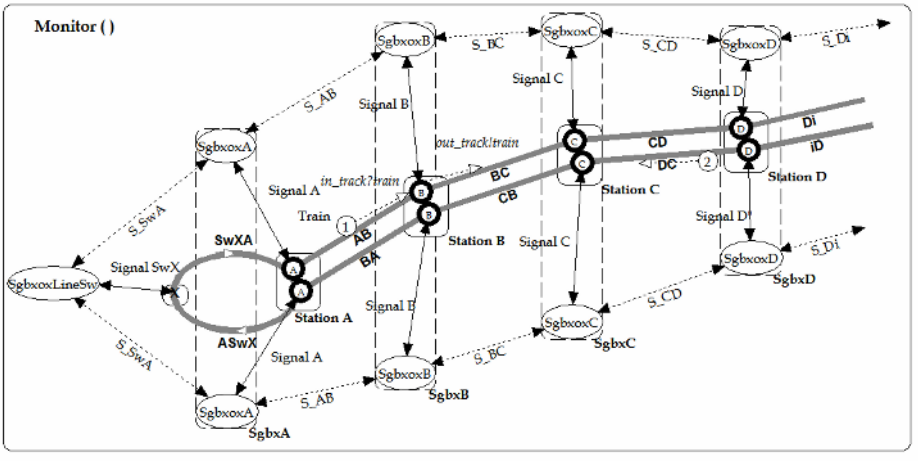


Fig. 8. Operational view of DSBs based interlocking control applied to a line segment

```
proctype Monitor()
{
do
 :: assert(nfull(TunnelSwA)&&nfull(TunnelAB)&&nfull(TunnelBC)
&&nfull(TunnelCD)&&nfull(TunnelDE)&&nfull(TunnelEF)
&&nfull(TunnelFG)&&nfull(TunnelGH)&&nfull(TunnelHI)
&&nfull(TunnelIJ)&&nfull(TunnelJK)&&nfull(TunnelKL)
&&nfull(TunnelLM)&&nfull(TunnelMN)&&nfull(TunnelNO)
&&nfull(TunnelOP)&&nfull(TunnelPQ)&&nfull(TunnelQR)
&&nfull(TunnelRS)&&nfull(TunnelST)&&nfull(TunnelTU)
&&nfull(TunnelUV)&&nfull(TunnelVW)&&nfull(TunnelWX)
&&nfull(TunnelXSw1)&&nfull(TunnelSw1X)&&nfull(TunnelXW)
&&nfull(TunnelWV)&&nfull(TunnelVU)&&nfull(TunnelUT)
&&nfull(TunnelTS)&&nfull(TunnelSR)&&nfull(TunnelRQ)
&&nfull(TunnelQP)&&nfull(TunnelPO)&&nfull(TunnelON)
&&nfull(TunnelNM)&&nfull(TunnelML)&&nfull(TunnelLK)
&&nfull(TunnelKJ)&&nfull(TunnelJI)&&nfull(TunnelIH)
&&nfull(TunnelHG)&&nfull(TunnelGF)&&nfull(TunnelFE)
&&nfull(TunnelED)&&nfull(TunnelDC)&&nfull(TunnelCB)
&&nfull(TunnelBA)&&nfull(TunnelASw))
od
}
```

Fig. 9. Safety assertion ASR1

In Line 1 of the Athens underground metro network, due to its track layout we expect that each train loops through the line's interlocking points (stations) and there are no unreachable stations.

To prove this expectation we verify that in an infinite run the train eventually passes through declared tunnel routes (and corresponding stations).

```
Full statespace search for:
never claim         - (not selected)
assertion violations+
cycle checks        - (disabled by -DSAFETY)
invalid end states- (disabled by -E flag)

State-vector 1588 byte, depth reached 2189, errors: 0
  141241 states, stored
  280500 states, matched
  421741 transitions (= stored+matched)
     102 atomic steps
hash conflicts: 20022 (resolved)

Stats on memory usage (in Megabytes):
225.421 equivalent memory usage for states (stored*(State-vector + overhead))
132.292 actual memory usage for states (compression: 58.69%)
State-vector as stored = 929 byte + 8 byte overhead
2.097 memory used for hash table (-w19)
0.320 memory used for DFS stack (-m10000)
0.605 memory lost to fragmentation
134.104 total actual memory usage
```

Fig. 10. Full state space results for assertion ASR1

This requirement is specified by the following liveness property:

LVN1: "*For any tunnel route, as soon as a train loops in its line, this train will eventually pass through it*"

We express the forenamed property by the following LTL (Linear Temporal Logic) formula

```
[] (<> p) -> (<> q) -> (<> p)
```

with the symbol definitions

```
#define p (len(TunnelAB)==0)
#define q (len(TunnelAB)==1)
```

and the used temporal operators defined as follows:

`<>x = TRUE U x`	*eventually*
`[] x = ¬ <> ¬x`	*always*
`->`	*logical implication*

The recurrence formula `[](<> p)` asserts that in an infinite sequence of states `p` occurs infinitely many times. If so, the train should eventually pass through the checked tunnel route. SPIN generates the never claim (finite automaton shown in Figure 11) of the *LVN1* formula and checks if the expected property holds for all executions. As a result we get that *LVN1* is valid.

```
never {    /* !(( []  (<>  p ) -> (<>  q) -> (<>  p))) */
T0_init:
if
:: (! ((p))) -> goto accept_S4
:: (! ((p))) -> goto accept_S7
fi;
accept_S4:
if
:: (! ((p))) -> goto accept_S4
fi;
accept_S7:
if
:: (! ((p))) -> goto accept_S4
:: (! ((p))) -> goto T0_init
fi;
}
```

Fig. 11. Never claim for the *LVN1* formula

5 Conclusion

In this paper we introduced an interlocking control that is based on a peer-to-peer signaling approach, between the so-called Distributed Signal Boxes and the network's interlocking points. Compared to all other centralized solutions, the proposed interlocking control does not depend on proprietary concept definitions that refer to railway interlocking systems, invests on design simplicity, provides improved scalability prospects and avoids the problem of the single point of failure.

Our control design is a general signaling technique that is possible to be applied in a wide range of control problems. We describe its application to the topology of the Athens underground metro network, but we have also verified its validity in networks that include a second type of DSB communication link and more precisely the 1-to-2-split communication link shown in Figure 1. DSBs correspond to the network's interlocking points and communicate only with them, as well as with their neighbor signal boxes. We proved that the proposed interlocking control averts the possibility of collision between two trains in one of the network's tracks and that each train will eventually pass through all tracks (and stations) of the line, where the trains move. Model checking was performed based on the use of the SPIN model checker. We noted that when the proposed control design is applied to a real-scale problem, compared to the published centralized solutions results in a smaller and easier to analyze state space.

As a first priority future research prospect we consider the introduction of a third signal exchanged between the network's interlocking points and DSBs. This will allow us to cover one more case of communication link and more precisely the 2-to-1-join communication link that is not tested so far. We also consider extending the proposed interlocking control, such as to include triple-modular redundant DSBs. In this way, we aim to develop a complete fail-safe control design, based on the signaling technique that we described in this paper.

References

[1] Hansen, K. M., Formalizing Railway Interlocking Systems, In Proceedings of the 2nd FMERail Workshop, 1998

[2] Garmhausen, V. H., Campos, S., Cimatti, A., Clarke, E., Giunchiglia, F., Verification of a Safety-Critical Railway Interlocking System with Real time Constraints, Science of Computer Programming, Vol. 36, No. 1, Elsevier North-Holland, 53-64, 2000

[3] Campos, S., Clarke, E., Minea, M., The Verus tool: a quantitative approach to the formal verification of real-time systems, In Proceedings of the Conference on Computer Aided Verification, 1997

[4] Cimatti, A., Giuchiglia, F., Mongardi, G., Romano, D., Torielli, F., Traverso, P., Model Checking Safety Critical Software with SPIN: an Application to a Railway Interlocking System. In Proceedings of the 3rd SPIN workshop, 1997

[5] Cimatti, A., Giunchiglia, F., Mongardi, G., Romano, D., Torielli, F., Traverso, P., Formal Verification of a Railway Interlocking System using Model Checking, Formal Aspect of Computing, Vol.10, No. 4, 361–380, 1998

[6] Holzmann, G. J., The Model Checker SPIN, IEEE Transaction on Software Engineering, Vol.5, No.23, 279–295, 1997

[7] Gnesi, S., Latella, D., Lenzini, G., Abbaneo, C., Amendola, A., Marmo, P., A Formal Specification and Validation of a Critical System in Presence of Byzantine Errors, In Proceedings of TACAS 2000, Lecture Notes in Computer Science 1785, 535–549, 2000

[8] Winter, K., Robinson, N. J., Modelling large railway interlockings and model checking small ones, In Proceedings of the 26th AustralAsian Computer Science Conference in Research and Practice in Information Technology, Adelaide, Australia, 309-316, 2003

[9] Hlavaty, T., Preucil, L., Stepan, P., Klapka, S., Formal methods in development and testing of railway interlocking systems, In Proceedings of the Conference on Intelligent Methods for Quality Improvement in Industrial Practice. Prague: CTU FEE, Department of Cybernetics, The Gerstner Laboratory, Vol. 1, 14-25, 2002

[10] Haxthausen, A. E., Peleska, J., Formal Development and Verification of a Distributed Railway Control System, IEEE Transactions on Software Engineering, Vol. 26, No. 8, 687-701, 2000

[11] Arozarena, P., Frints, M., Collins, S., Fallon, L., Zach, M., Serrat, J., Nielsen, J., Madeira: A peer-to-peer approach to network management, In Proceedings of the Wireless World Research Forum, Shanghai, China, April 2006

[12] The SPIN model checker official website, available at http://spinroot.com/

[13] Holzmann, G. J., Design and Validation of Computer Protocols, Prentice-Hall, 1991.

A Delay Time-Based Peak Load Control for Stable Performance

Yonghwan Lee[1], Eunmi Choi[2,*], and Dugki Min[1,**]

[1] School of Computer Science and Engineering, Konkuk University,
Hwayang-dong, Kwangjin-gu, Seoul, 133-701, Korea
{yhlee, dkmin}@konkuk.ac.kr
[2] School of Business IT, Kookmin University
Chongung-dong, Songbuk-gu, Seoul, 136-702, Korea
emchoi@kookmin.ac.kr

Abstract. Currently, integration application, such as EAI (Enterprise Application Integration), B2BI (Business-To-Business Integration), requires reliable B2B integration framework that can stably process massive I/O transactions during overload state. This paper proposes the delay time-based peak load control for stable performance and we describe that how the suggested pattern is applied to B2BI as an example. The pattern uses the delay time algorithm for controlling the heavy peak load caused by many requests for a short period of time. According to our experimental result, the proposed delay time algorithm can stably process the heavy load after the saturation point and has an effect on the controlling the peak loads.

1 Introduction

The evolution of internet has brought a new way for enterprises to interact with their partners. Many infrastructures and enterprise information systems have been developed to extend business and value-added services on the Internet. Since markets are rapidly changing, business partners tend to be changed dynamically and system instability gradually tends to increases due to service congestion for a short period of time [1].

A distributed clustering and hardware extension has been proposed for solving system instability [2]. However, in view of cost-effective aspect, it is not good way to pay many expenses to the systems at which the service congestion phenomenon happens one or two a year. For solving the temporary service congestion with reasonable expenses, we can use the On-Demand services provided by many hardware vendors. The On-Demand services can be good choice in view of ratable charge of service usage. However, those services have the difficulty of manageability and tracking of

* This work was supported by research program and the research center UICRC of Kookmin University in 2006.
** Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 329 – 340, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the cause of many faults. Moreover, as the price charge of the On-Demand services is based on the number of CPU, there exists the weak point of price rise.

A PLC (Peak Load Control) is the mechanism for preventing thrashing in transaction processing system by controlling the number of concurrently running transactions. The term thrashing generally describes a phenomenon where an increase of the load results in decrease of throughput (or another-related performance measure) [3]. Usually, we can distinguish a load-throughput function into three phases [3]: underload, saturation, and overload. The underload phase is the state of light loads with sufficient resources available and the throughput grows almost linearly making use of possible parallelism in the system. The saturation phase is the state that reaches highest throughput. When the finite capacity of the system becomes effective, the throughput function flattens out. After the saturation phase, further increasing the load will not lead to an asymptotic approach to the saturation bound but will cause a sometimes sudden drop in throughput. In other word, the phenomenon of thrashing happens at overload phase.

In this paper, we propose the Worker-Linker pattern that can effectively process massive I/O transactions and control a heavy overload caused by request congestion for a short period of time. The proposed pattern adopts an I/O processing-related effectiveness from concurrent-related patterns (e.g. Worker pattern [4], Connector-Acceptor pattern [5], Reactor pattern [6], and Proactor pattern [7]) and the scalability and flexibility of business logics from command pattern. In addition to adaptation of existing concurrent pattern, the proposed pattern adds the PLC mechanism to the existing Worker pattern for controlling the heavy overload caused by request congestion [8]. This pattern uses the delay time algorithm for the PLC mechanism. This paper also shows the example of applying the pattern to business-business integration framework and the experimental result for proving the stability of performance. According to our experiment result, the proposed delay time algorithm can stably control the heavy overload after the saturation point and has an effect on the controlling peak load.

The remainder of this paper is organized as follows. In section 2, we survey the existing patterns adapted by the Worker-Linker pattern and load control methods. Section 3 describes the proposed Worker-Linker pattern. Chapter 4 presents the example of applying the pattern to business-business integration framework. Section 5 shows the experimental results for proving the stability of performance. Section 6 draws a conclusion.

2 Related Works

There are many patterns for effectively processing massive I/O transactions. The Acceptor-Connector design pattern [9] decouples connection establishment and service initialization in a distributed system from the data communication performed after a service initialization. The Acceptor-Connector pattern enhances the reusability, portability, and extensibility of connection-oriented software and also efficiently utilizes the inherent parallelism in the network and hosts.

The Acceptor-Connector pattern is categorized into the Reactor and the Proactor pattern according to the method of event dimultiplex. In the Reactor pattern, after the *Acceptor* registers event handler for processing events, the Acceptor waits for one or more events on a set of handles. In other words, the Reactor pattern uses the synchronous multiplexer for demultiplexing and dispatching of multiple event handlers. In contrast to the Reactor pattern, the Proactor pattern supports the demultiplexing and dispatching of multiple event handlers, which are triggered by the completion of asynchronous events. This pattern simplifies asynchronous application development by integrating the demultiplexing of completion events and the dispatching of their corresponding event handlers. The Proactor pattern generally can support the higher throughput than the Reactor pattern due to little consumption of resources. However, the Proactor pattern can increase the complexity and instability of a system due to the synchronization of the shared resources.

Generally, the threads for concurrent processing can be categorized into the following two threads [10]: actor-based threads, task-based threads. The actor-based threads mainly focus in interaction with another actor in response to external events. In contrast with the actor-based threads, the task-based threads generally have the purpose of efficient job processing instead of interaction with another actor. The Worker thread pattern is the kind of the task-based thread and the Producer-Consumer pattern [11].

The Worker thread pattern is generally based on the following concept; When request events arrives, putting the received events to a queue and asynchronously processing the events by another threads is faster than directly processing the received events by creating new threads. The pattern can also improve the structure of some task-based concurrent programs by allowing you to package many smaller, logically asynchronous units of execution as tasks. Moreover, the pattern can prevent the resource exhaustion and the context switching overhead due to restriction on the number of Worker threads. The explicit queuing of the Worker pattern also permits greater flexibility in tuning execution semantics (e.g. priority of events).

The maximum number of concurrent Worker threads is a system parameter that is tuned by the system administrator when the system is installed or started up. When the transaction load is constant and the value is chosen appropriately, this solution may work. However, traces from real systems show large variations of the load, both quantitative and qualitative. For solving this problem, many adaptive load control mechanisms [12] have been proposed. In contrast with existing load control mechanisms, our approach does not control the number of concurrent Worker threads but the delay time of each active thread for solving thrashing phenomenon. Moreover, our approach adds the load control mechanism to the existing Worker pattern.

3 The Worker-Linker Pattern

Currently, I/O-based transaction systems, such as EAI, B2Bi, and ESB (Enterprise Service bus), especially require the effective and reliable processing of massive I/O-based transactions. System instability gradually tends to increases due to service

congestion for a short period of time. This chapter describes the Worker-Linker pattern for solving those problems. The proposed pattern is constructed as a pair of the Worker and Linker like the Acceptor-Connector pattern. So, the pattern helps a software designer to easily design complex I/O-related modules. The pattern also adds the PLC mechanism to the effectiveness of the existing Worker thread pattern so that it can not only provide effective processing of massive I/O-based transactions but also remove the system instability caused by request congestion for a short period of time. Moreover, the pattern uses the command pattern for executing extensible and flexible business logics. Figure 1 shows the sequence diagram of the Worker-Linker pattern.

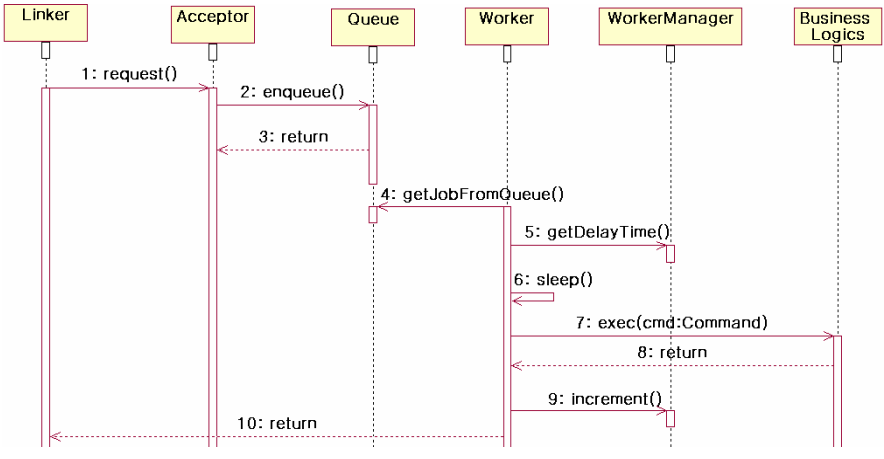


Fig. 1. A Sequence Diagram of the Worker-Linker Pattern

The *Linker* is similar to the *Connector* of the Acceptor-Connector pattern. After the *Acceptor* receives messages from the *Linker*, it puts them a queue and wake up a waiting *Worker* thread. The *Worker* thread gets them from the queue and gets the delay time from the WorkerManager. After the *Worker* thread sleeps for the delay time calculated by the *WorkerManager*, the *Worker* thread executes business logics by using a command pattern. Finally, the *Worker* increments the number of transactions processed by the Worker threads.

Figure 2 is the activity diagram of the *Worker* threads. The *Worker* thread locks a queue and gets jobs from it. If there are no jobs in the queue, the *Worker* thread goes into the waiting queue of the *WorkerManager*. When an *Acceptor* gets request messages from a *Linker*, it wakes up the waiting *Worker* thread in the *WorkerManager*. If there are jobs to process, The *Worker* thread gets the delay time calculated by the *WorkerManager* and sleeps for the delay time. After sleeping, the *Worker* thread processes the jobs and finally increments the number of transactions processed by the Worker threads.

Figure 3 is the pseudo code for the *WorkerManager*'s delay time algorithm. After sleeping during a check interval time, the *WorkerManager* gets the number of transactions processed by all Worker threads and the maximum transaction process speed

configured by a system administrator. And then, the WorkerManager calculates the TPMS (Transaction per Milliseconds) by dividing the number of transactions by the maximum transaction processing speed and calculate the over speed between the TPMS and the maximum transaction processing speed. If the value of the over speed is greater than zero, the system is considered as an overload state. Accordingly, it is necessary to control the overload state. On the contrary, if the value of the over speed is zero or less than zero, it is not necessary to control the speed of transaction processing.

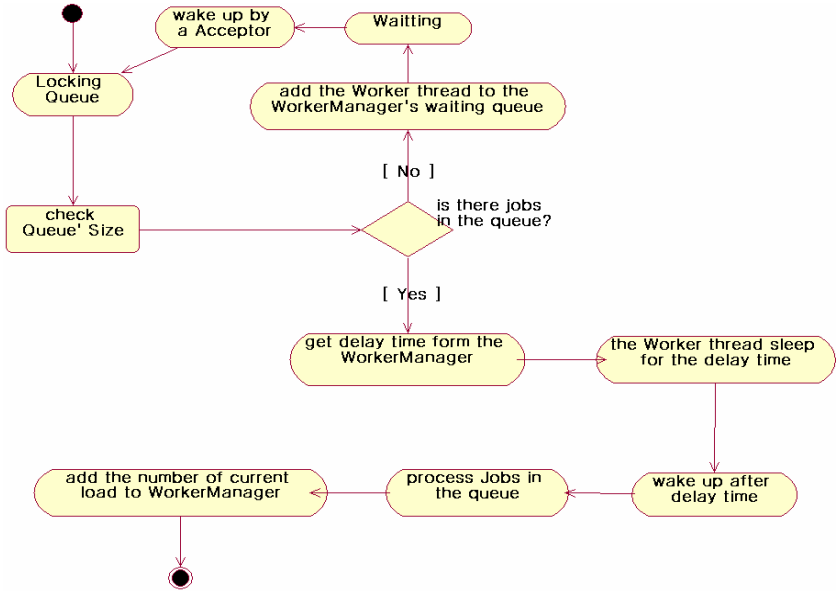


Fig. 2. An Activity Diagram of the Worker Thread

For controlling the overload state, this paper proposes the delay time algorithm for making each Worker threads sleeping for the delay time. $OS(t_{i+1})$ is the over speed between the transaction processing speed ($TPMS(t_{i+1})$) at the time t_{i+1} and the configured maximum transaction processing speed (MTPS). The $OS(t_{i+1})$ is calculated by applying the formula (1). If the over speed $OS(t_{i+1})$ is greater than zero, the formula (2) is used for getting a new delay time $D(t_{i+1})$ at the time t_{i+1}. The $N(t_{i+1})$ of the formula (2) means the number of active *Worker* threads at the time t_{i+1} and $D(t_i)$ means the delay time at the time t_i. If the $D(t_i)$ is zero, $D(t_i)$ must be set zero.

If the $OS(t_{i+1})$ of the formula (1) is zero or less than zero, a new delay time $D(t_{i+1})$ at the time t_{i+1} is differently calculated according to the delay time $D(t_i)$ at the time t_i. If the $D(t_i)$ at the time t_i is greater than the criterion value δ, the $D(t_{i+1})$ is calculated by applying the formula (3). On the contrary, if the $D(t_i)$ is same or less than the criterion value, $D(t_{i+1})$ is set zero.

The criterion value is used for preventing repetitive creation of the over speed generated by directly setting the delay time from the top to zero. When the system state is

continuously in state of heavy load for a short period of time, it tends to regenerate the over speed to increment the much delay time at the time t_i and then directly set the delay time zero at the time t_{i+1}. The criterion value is the previous delay time that can decide whether next delay time is directly set zero or not. The βpercent of the formula (3) decides the slope of a downward curve. However, if the delay time at the time t_i is lower than the criterion value. The new delay time at the time t_{i+1} is set zero. Accordingly, When a system state changes from the heavy overload at the time t_i to the underload at the time $t_{i+1,}$ The gradual decrement by The βpercent prevents the generation of repetitive over speed caused by abrupt decrement of the next delay time. In order to implementing the PLC, All concrete Worker threads extend the PLCWorker thread class for adding the PLC mechanism.

```
1.while run_flag equals "true" do
2.  get interval time for checking load
3.  sleep for the interval time
4.  get the number of transactions processed during the interval time
5.  get the configured maximum speed
6.  TPMS := number of transactions / interval time
7.  over speed := TPMS – maximum speed
8.  If over speed >0 then
      8.1 get the previous delay time
      8.2 if previous delay time = 0
       8.2.1 previous delay time := 1
      8.3 get number of active worker thread
      8.4 new delay time:= over speed / number of active worker * previous delay time
9   else
      9.1 get current delay
      9.2 if current delay > δ
        9.2.1 new delay time := current delay * β
      9.3 else
        9.3.1 new delay time := 0
      9.4 end if
10. end if
11.end while
```

Fig. 3. A Pseudo Code for the WorkeManager's Delay Time Algorithm

$$OS(t_{i+1}) = TPMS(t_{i+1}) - MTPS \tag{1}$$

$$D(t_{i+1}) = OS(t_{i+1}) / N(t_{i+1}) * D(t_i) \tag{2}$$

$$D(t_{i+1}) = D(t_i) * \beta \tag{3}$$

$$D(t_{i+1}) = 0 \tag{4}$$

4 Applying to the Business-to-Business Integration Framework

This chapter describes the example of applying the Worker-Linker pattern to a B2Bi framework. The B2Bi framework takes the responsibility for processing trustworthy

integration transactions between source systems and target systems. Figure 4 shows the software architecture of B2Bi framework for applying the pattern.

The *Gateway* is comprised of the *Adapter*, the *Message Controller*, the *Dispatcher*, and the *Routing Manager*. The *Adapter* is in charge of processing communication-related details of sources systems or target systems, such as connection, listening, and conversion of protocols. The *Message Controller* takes the responsibility for controlling the flow of messages. If the *Message Controller* receives messages from the source adapters or target adapters, it decides the next destination of those messages based on the configurations for each tasks in the *Admin Console*. If business logics (i.e., data format transformation, data validation, etc) are configured for a specific task, the *Message Controller* invokes the *Dispatcher* to perform the business logics before or after communication with the target systems. To process business logics, the *Dispatcher* gets UTL components from the *UTL Processor* and executes them. The UTL component executes I/O-related business logics with a command object. The *UTL Processor* is in charge of creating and caching UTL components. If a UTL component does not exist in the UTL Processor, the *UTL Processor* gets the related information for creating UTL components from the *UTL Server* and then creates UTL components. An UTL is an xml-based language for defining data formats of header, request, and response block in a message.

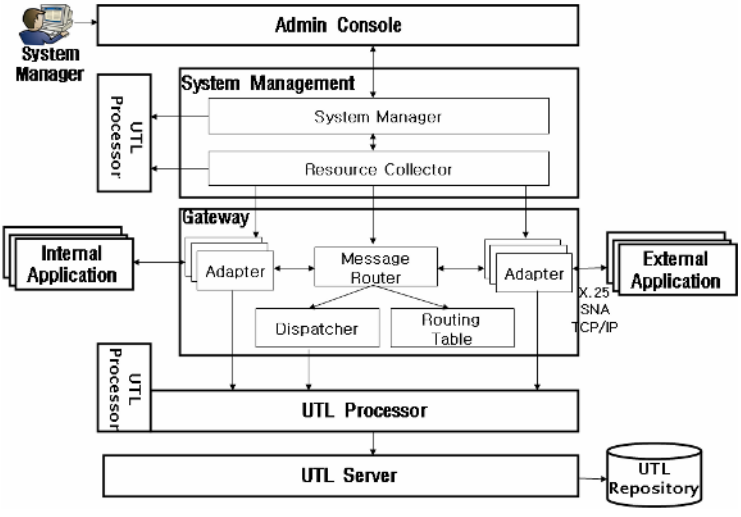


Fig. 4. Software Architecture of B2Bi Framework

The *Admin Console* has a development and a management tool. In view of a development tool, the *Admin Console* is in charge of managing source codes and various parameters according to the concerned task unit. In other word, the *Admin Console* categories tasks in a tree form, provides the related source code (i.e., UTL Code) to the task, edits them, compiles them, packages them, distributes them, and tests them

via a tool. Also the *Admin Console* manages setting information which is necessary for each task. In view of management tool, the Admin Console manages all resources related to the framework and monitors them.

The *System Management* is comprised of the *System Manager* and the *Resource Collector*. The *System Manager* is in charge of scheduling and managing the framework-related resources. For improving the qualities like performance and scalability, main modules (i.e., The *Adapter*, The *Gateway*, and The *Dispatcher*) may be deployed to different distributed system. The each main module has a thread called the *State Manager*. The thread gets resources-related information (i.e., process state, CPU, memory, I/O, etc) and sends them to the *Resource Collector*. The *Resource Collector* collects all resource information from the main modules and sends to the *System Manager*. The *System Manager* uses them for balancing load of each main module. Figure 5 is the example of applying the PLC mechanism of the Worker-Linker pattern to the B2Bi framework.

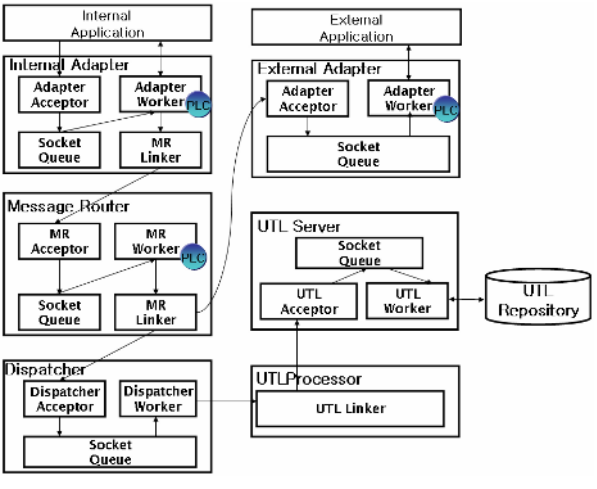


Fig. 5. Applying the Worker-Linker Pattern for Effectiveness and Stability of B2Bi Framework

Many modules (e.g. the *Adapter*, *Message Router*, *Dispatcher*, and *UTL Server*) of the B2Bi framework are designed with the sub modules, such as the *Worker*, *Linker*, and *Acceptor*. After the *Acceptor* receives the connection requests from the *Linker*, it put them into the *Socket Queue*. The *Worker* thread gets them from the *Socket Queue* and executes the business logics with a UTL-based command pattern. The *Linker* and *Acceptor* are responsible for controlling communication connections. The *Linker* and *Worker* are in charge of transmitting and receiving data. The *Linker* is responsible for connecting another module. Especially, the *MRLinker* in *Message Router* uses the routing table for connecting another module. The PLC mechanism of the Worker-Linker pattern is applied to the *Internal Adapter*, the *External Adapter*, and the *Message Router* and the *Worker* threads of those modules extend the *PLCWorker* object. Before the *Worker* thread executes the business logics with a command object, the

Worker thread gets the delay time from the *WorkerManager* and sleeps for the delay time calculated by the *WorkerManager*. The B2Bi framework can control the heavy load caused by request congestion.

5 Performance Analyses of the Worker-Linker Pattern

Currently, we apply the B2Bi framework of the chapter 3 to the 'k' bank of Korea. It can process 500 million transactions a day on the average. Moreover, it can support stable performance regardless of the heavy overload caused by request congestion. In this chapter, we show the performance analysis of the Worker-Linker pattern. As for load generation, the LoadRunner 8.0 tool [12] is employed. TPS (Transactions per Seconds) is used as a metric for performance analysis. As a purpose of performance experiment is not comparison with other integration system but the stability of the Worker-Linker pattern, we use one banking application like deposit ledger inquiry as a workload. The maximum speed, δand βfor delay time algorithm are configured 388, 100ms, and 0.75 respectively. Load Generator request to a JSP page and then the JSP page connects to the integration system with TCP protocol. The integration system gets data from IBM main frame with SNA protocol [13]. The date size received from IBM host is 475 bytes. The number of concurrent users is 300 and the value of think time is 0. The maximum TPS of the proposed integration system is 450 TPS (Transaction per Seconds) and average TPS is 381.

The B2Bi framework is designed by an existing Worker pattern for efficiency I/O. However, As the Worker pattern does not use the PLC mechanism. It has the tendency of instability of performance caused by heavy overload. To solve this problem, the B2Bi framework applies the PLC mechanism to the Worker-Linker pattern so that the number of Worker thread does not exceed maximum number configured by a system manager. Figure 6 is comparison of stability of performance between the Worker pattern and the Worker-Linker pattern.

The Worker pattern does not use the PLC mechanism but the Worker-Linker pattern uses the mechanism. Until concurrent user 300, the Worker-Linker pattern is similar to the Worker pattern in view of TPS and the maximum TPS of both patterns is 388. However, as the number of concurrent users is more than 300 users, both pattern show different symptoms. The Worker-Linker pattern holds 386 TPS on the average due to the PLC mechanism. However, the Worker pattern goes down in TPS until 550 concurrent users. After more than 550 users, the Worker pattern holds 120 TPS. Figure 7 explains the reason of different symptoms between the Worker pattern and the Worker-Linker pattern.

As the number of concurrent users is more than 300 users, the Worker pattern is in state of so heavy overload as to reach 100 percent CPU usage. However, because the Worker-Linker pattern has the PLC mechanism, the pattern holds 92 percent CPU usage. The Worker-Linker pattern adds the PLC mechanism to an existing Worker pattern with I/O efficiency so that it can control the instability of performance caused by request congestion.

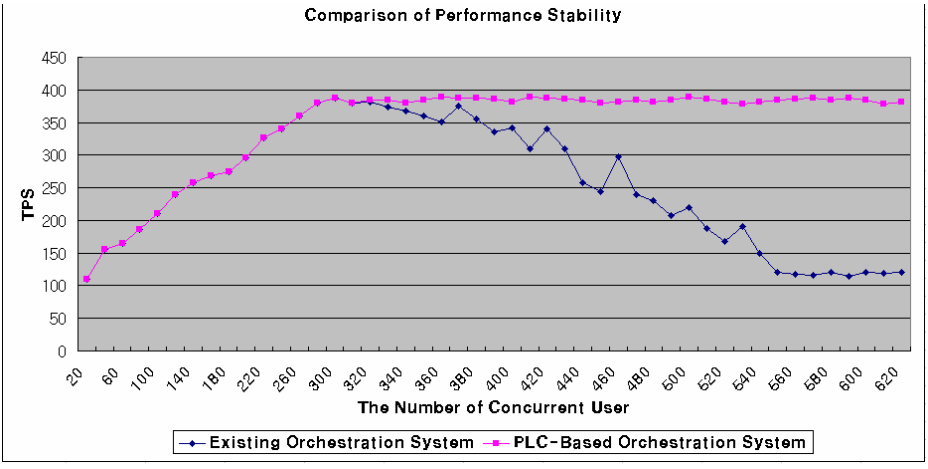


Fig. 6. Comparison of Performance Stability between the Worker and the Worker-Linker Pattern

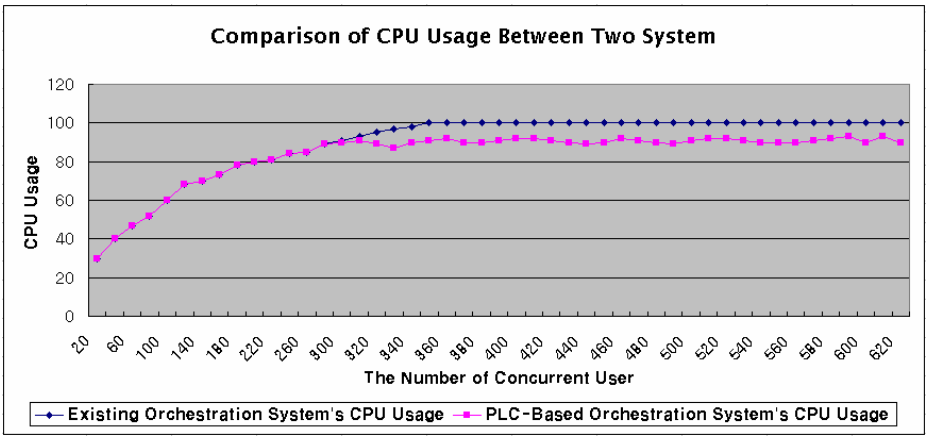


Fig. 7. Comparison of CPU Usage between the Worker Pattern and Worker-Linker Pattern

Figure 8 show the relationship between the over speed and the delay time after the saturation point. This experimental result proves that the proposed delay time algorithm of the WorkerManager has an effect on controlling the over speed. As the number of concurrent users is more than 300 users, the over speed frequently happens. Whenever the over speed happens, each Worker thread sleeps for the delay time calculated by the WorkerManager. As the higher over speed happens, each Worker thread sleeps for the more time so that the over speed steeply goes down. Although the over speed steeply goes down, the delay time does not steeply goes down due to the criterion value δ. As the criterion value is set 100 ms in this experiment, the delay time gradually goes down until the 100 ms. As soon as the delay time passes 100 ms, the next delay time is set zero. Figure 8 shows that the over speed does not happen

from top delay time to zero delay time. As soon as the delay time passes zero, the over speed again happens.

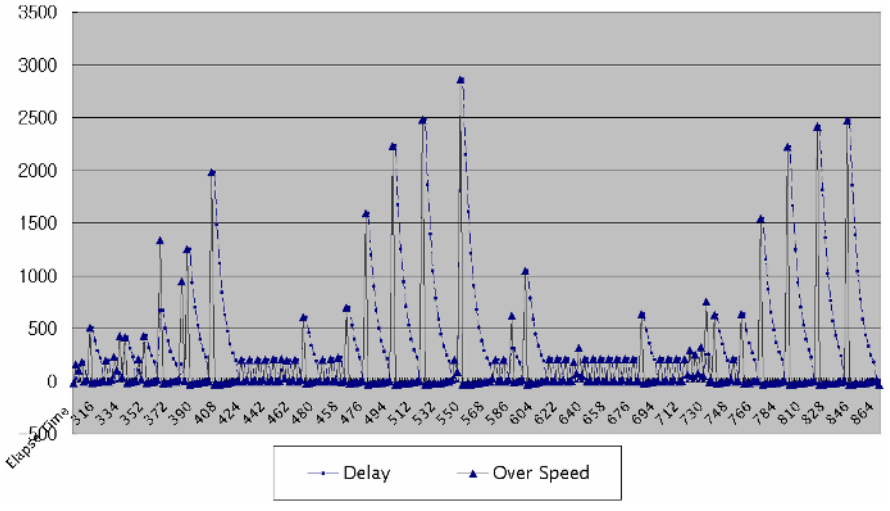


Fig. 8. Control of Over Speed by the Delay Time Algorithm

6 Conclusions

I/O-driven integration applications, such as EAI, B2Bi, home gateways, and heterogeneous devices integration, need a trustworthy framework to process a large volume of I/O-driven transactions with high performance and reliability. This paper describes the Worker-Linker pattern for a large volume of transactions. The Worker-Linker pattern adds the PLC mechanism to an existing Worker pattern with I/O efficiency so that it can control the instability of performance caused by request congestion. This paper uses the delay time algorithm for control the peak load caused by heavy request in a short time period. According to our experiment result, the proposed delay time algorithm can stably process the heavy load after the saturation point and has an effect on the controlling peak load.

References

[1] I. Ahmad and A. Ghafoor, “Semi-Distributed Load Balancing for Massively Parallel Multicomputer Systems,” IEEE Trans. Software Eng., vol. 17, no. 10, pp. 987-1004, Oct. 1991.

[2] O.P. Damani et al., One-IP: techniques for hosting a service on a cluster machines, J. Coputer Networks and ISDN Systems, Vol. 29, Elsevier Science, Amsterdam, Netherlands, Sept. 1997, pp. 1,019-1,027.

[3] P. J. Denning: Thrashing: Its Causes and Prevention. Proc. AFlPS FJCC 33, 1968, pp, 915-922

[4] Robert Steinke, Micah Clark, Elihu Mcmahon, "A new pattern for flexible worker threads with in-place consumption message queues", Volume 39 , Issue 2 (April 2005) table of contents Pages: 71 - 73 Year of Publication: 2005.
[5] D. C. Schmidt, "Acceptor and Connector: Design Patterns for Initializing Communication Services," in Pattern Languages of Program Design (R. Martin, F. Buschmann, and D. Riehle, eds.), Reading, MA: Addison-Wesley, 1997.
[6] D. C. Schmidt, "Reactor: An Object Behavioral Pattern for Concurrent Event Demultiplexing and Event Handler Dispatching," in Pattern Languages of Program Design (J. O. Coplien and D. C. Schmidt, eds.), pp. 529–545, Reading, MA: Addison-Wesley, 1995
[7] J. Hu, I. Pyarali, and D. C. Schmidt, "Applying the Proactor Pattern to High-Performance Web Servers," in Proceedings of the 10th International Conference on Parallel and Distributed Computing and Systems, IASTED, Oct. 1998.
[8] Performance Stability. http://www.performance-stability.com/
[9] Douglas C. Schmidt, Michael Stal, Hans Rohert, and Frank Buschmann, "Pattern-Oriented Software Architecture: Concurrent and Networked Objects", John Wiley and Sons, 2000.
[10] Doug Lea, Concurrent Programming in Java, Second Edition, Addison-Wesley, November, 1999
[11] [9] R. G. Lavender and D. C. Schmidt, "Active Object: an Object Behavioral Pattern for Concurrent Programming," in Proceedings of the 2nd Annual Conference on the Pattern Languages of Programs, (Monticello, Illinois), pp. 1–7, September 1995.
[12] Hans-Ulrich Heiss, Roger Wanger, "Adaptive Load Control in Transaction Processing Systems", Proceedings of the 17th International Conference on Very Large Data Bases, September 1991

Interference Aware Dynamic Subchannel Allocation in a Multi-cellular OFDMA System Based on Traffic Situation

Banani Roy[1], Chanchal Kumer Roy[1], and Michael Einhaus[2]

[1] Queen's University, Kingston, Ontario, Canada K7L3N6
{broy, croy}@cs.queensu.ca
[2] RWTH Aachen University, Aachen, Germany

Abstract. This paper presents the development and evaluation of a dynamic subchannel allocation scheme for downlink multi-cellular Orthogonal Frequency Division Multiple Access (OFDMA) systems. In the considered system each Access Point (AP) and the associated Mobile Terminals (MTs) are not operating on a frequency channel with a fixed bandwidth, rather the channel bandwidth for each AP is dynamically adapted according to the traffic load. The subchannels assignment procedure is based on quality estimations due to the interference measurements and the current traffic load. The developed dynamic subchannel allocation ensures Quality of Service (QoS), better traffic adaptability and higher spectrum efficiency with less computational complexity.

1 Introduction

Orthogonal Frequency Division Multiplexing (OFDM) is a multiple carrier technique that has been proved to be robust for communication over fading channels. OFDM systems divide a broadband channel into many narrowband orthogonal subcarriers [8], [3] where a group of subcarriers forms subchannels. When multiple accesses are desired OFDM can be combined with Frequency Division Multiple Access (FDMA) or Time Division Multiple Access (TDMA) or a mix of both. The combination of OFDM and TDMA is called Orthogonal Frequency Division Multiple Access (OFDMA). OFDMA is being considered as a modulation and multiple access method for the 4^{th} generation wireless networks. In OFDMA systems, radio resource allocation is still a critical issue in optimizing the performance of the system since the rapid increase in the size of the wireless mobile community and its demand for high speed multimedia communications stand in clear contrast to limited spectrum resources that have been allocated in international agreements. A feasible or nearly optimal dynamic subchannel allocation technique can be used for the resource allocation of an OFDMA system in order to optimize the performance. The motivation of this work is to allocate OFDMA subchannels to APs in a centrally controlled manner using Dynamic subChannel Allocation (DCA) scheme with effective frequency reuse by removing the inter cell interference remarkably. The newly developed DCA scheme works better than the Fixed subChannel Allocation (FCA) scheme with

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 341–352, 2006.
© Springer-Verlag Berlin Heidelberg 2006

optimal Fixed Reuse Partitioning (FRP) [6] and the complexity of the algorithm is $O(n^2 \cdot m) + O(k \cdot n^2)$, n is number of cells, k is number of subchannels, m is the number of users in the system.

The remainder of the paper is organized as follows. In Section 2 some previous works related to DCA scheme in OFDMA systems are presented. Section 3 discusses the developed DCA scheme, whereas the algorithmic complexity of this scheme is analyzed in Section 4. Section 5 briefly describes the deployed OFDMA system. The simulation results obtained from this system is provided in Section 6 and finally, Section 7 concludes the paper with some potential guidelines for further enhancement of this work.

2 Previous Work

Several DCA algorithms are developed, but their complexities are rather high [6]. In [12], the best channel reuse pattern was selected by doing the capacity prediction for all possible combination of the channel patterns. However, this type of solution is in practice impossible, especially where the number of cells and channels are many. Furukawa et al. [5], have mentioned that despite of the absence of intra-cell interference in OFDMA systems, optimum channel allocation is still a NP-hard problem which is basically complex. Moreover, individual user's rate requirements further complicate the problem. Kim et al. [7] and Pietrzyk et al. [11], have developed a channel allocation scheme based on [2] with less computational complexity. But in [11], a lot of iterations are needed to solve the problem and the linearization in [7] cannot be generalized to all types of modulations. Yin et al. [14], have investigated non-iterative algorithms to reduce computational complexity.

3 Developed DCA Scheme

In the developed subchannel allocation scheme traffic load is estimated based on the utilization of the assigned subchannels and the interference estimation is done based on the received signal power measurements. For this, a Hierarchical Radio Resource Management scheme of an OFDMA system has been designed by introducing an Access Point Controller (APC). The APC aims to handle inter cell interference and to allocate subchannels to the APs dynamically. Its functionalities are divided into two phases: reuse decision phase and allocation phase. In the first phase, the APC gains knowledge about the interference situation by investigating the mutual Signal to Interference Ratio (SIR) of the APs. In the second phase, the APC allocates subchannels to the APs using the obtained knowledge of interference. Two phases are discussed as follows:

3.1 Reuse Decision Phase

We have used the concept of Self Organizing Reuse Partitioning (SORP) [6], [5], but in a resource optimizing and centralized manner by exploiting the received signal power of the broadcast bursts in order to form the adaptive co-channel

cells. This approach does not require any extra resources for calculating the mutual SIR. In this technique, the APC triggers the APs to measure the received signal power of the MTs periodically. After getting the invocation from the APC, the AP triggers its associated MTs to measure the received signal power level of the broadcast bursts. Then the associated MTs measure the received signal power level from all the APs and send the measured values to their controller AP. The controller APs then inform the APC about the received signal power values and the APC then estimates the mutual SIR of the APs using the received signal power values. Let us consider a small scenario with three APs: AP_0, AP_1 and AP_2. Each AP has four associated MTs. In this scenario the distance of AP_0 & AP_1 is 500m, AP_0 & AP_2 is 790m, AP_1 & AP_2 is 790m and each cell radius is 200m where cells are not overlapping. As AP_2 is far apart from AP_0 and AP_1, it can reuse resources of both AP_0 and AP_1. On the other hand, as AP_1 and AP_2 are not far apart, they have very low chance to reuse each other resources due to higher interference. The estimated mutual SIR values are shown in Table 1. The APC forms co-channel cells according to the mutual SIR values by comparing these with a predefined reuse threshold (here it is considered $15dB$) and stores the binary reuse decisions: *reusable(y)* and *notreusable(n)* in a table called Reuse Partitioning Table (RPT). Different Reuse Constraints (RCs) can be used to form co-channel cells such as min mutual SIR, mean mutual SIR and weighted mutual SIR. Both in the example and simulation results, we have used

Table 1. Mutual SIR values for three–cell scenario

	$MT_{0,i}/SIR$	AP_0/AP_0	AP_0/AP_1	AP_0/AP_2
AP_0	$MT_{0,100}$	- -	16.54	24.07
	$MT_{0,101}$	- -	13.54	25.25
	$MT_{0,102}$	- -	12.45	21.9
	$MT_{0,103}$	- -	18.46	29.41
	$MT_{1,i}/SIR$	AP_1/AP_0	AP_1/AP_1	AP_1/AP_2
AP_1	$MT_{1,104}$	16.13	- -	27.09
	$MT_{1,105}$	13.53	- -	25.63
	$MT_{1,106}$	12.39	- -	25.65
	$MT_{1,107}$	18.68	- -	25.64
	$MT_{2,i}/SIR$	AP_2/AP_0	AP_2/AP_1	AP_2/AP_2
AP_2	$MT_{2,108}$	24.07	24.43	- -
	$MT_{2,109}$	33.03	33.41	- -
	$MT_{2,110}$	45.49	45.69	- -
	$MT_{2,111}$	33.12	33.29	- -

min RC, the minimum mutual SIR for all the MTs associated to an AP (for details see [13]). But it is not possible to remove the interference remarkably by considering merely this binary decision due to the indirect resource sharing. It has been noticed that when two APs form co-channel cells, another AP could be interfered if it can only be able to form co-channel cells with one of the two APs.

The APC filters this type of indirect interference by finding out a conflicting set of APs where APs interfere each other by reusing other APs' resources. Then it updates the RPT by introducing another reuse decision: *partially reusable (p)* for those conflicting APs by replacing the reusable decision. For example in the above mentioned scenario, AP_0 & AP_2 and AP_1 & AP_2 can form co-channel cells, but AP_0 & AP_1 cannot by using min RC (see Table 1). Thereby, if AP_0 and AP_1 use same resources of AP_2, they will interfere with each other. Consequently, the APC partially assigns AP_2's subchannels to AP_0 and AP_1

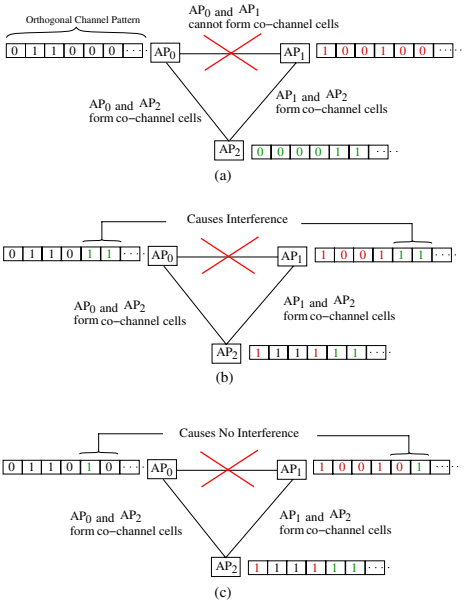


Fig. 1. Filtering interference

in an orthogonal manner which is shown in Fig. 1. The APC then assigns a weighted value to each decision where the *reusable* decision is assigned value 1 i.e. an AP can use whole resource of the interferer AP. The *notreusable* decision is assigned value 0 i.e. resources cannot be reused between two APs. The value for the *partial* decision is calculated based on the number of candidate APs using another AP's exclusive resources in an orthogonal manner; for example, in the scenario explained earlier, AP_1 and AP_2 will partially use AP_0's resources. As there are two APs to use AP_0's orthogonal subchannels, the value of the partial decision will be 1/2. The sum of each AP's decision values called Reuse Decision Values (RDVs) are shown in Fig. 2 (bottom–right part).

3.2 Allocation Phase

In this phase, the subchannel requirements of the APs are calculated by multiplying the value of the utilization function (weighted mean of the utilization

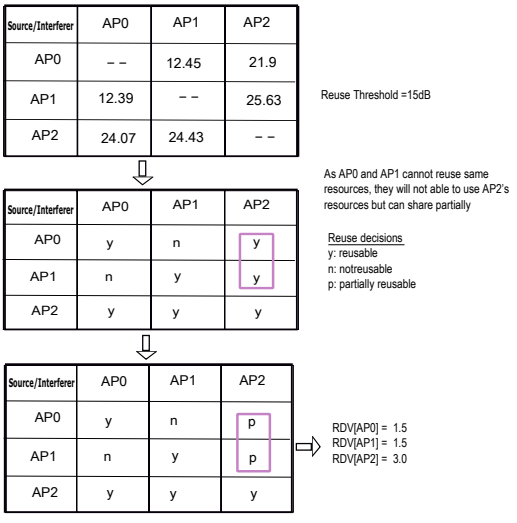


Fig. 2. Working steps of reuse decision phase

[12], [13]) with the previously assigned subchannels of APs. The higher utilization function value indicates higher resource requirement for an AP. Then the priority level of an AP is calculated for the assignment of orthogonal subchannels by dividing the subchannel requirements with the corresponding RDV. The steps in the reuse decision phase are as follows:

- Calculate the required subchannels using estimated utilization and previously assigned subchannels
- Loop until all the subchannels are assigned (allocation of orthogonal subchannels)
  - Calculate priority levels
  - Select the AP with the highest priority
  - Assign subchannel to the selected AP
  - Reduce the priority level
- Distribute reusable subchannels based on the RPT

4 Algorithm Complexity

The complexity to calculate mutual SIR is $O(n^2 \cdot m)$. Here two loops involve the n number of APs associated to the APC and one loop involves m number of MTs associated to an AP. The complexity of generating RDV matrix is $O(n^2)$ where two loops involve the n number of APs associated to the APC. The complexity of the orthogonal subchannel allocation scheme is $O(k \cdot n^2)$ as it involves three loops. The outer most loop involves k number of subchannels and the inner two loops involve n number of APs. The complexity of distributing reusable subchannels is $O(n^2 \cdot k)$ as it also involves two loops for the n number of APs

and one loop for k subchannels. So the over all complexity of this algorithm can be written as $O(n^2 \cdot m) + O(k \cdot n^2)$ which is better than previous works cited in Section 2.

5 Deplyed OFDMA System Description

The MAC structure is based on IEEE 802.16a, whereby the Physical Layer (PHY) differs significantly. The channel bandwidth of the system is 80MHz which is subdivided into 1024 subcarriers with a spacing of 78.125kHz. Due to orthogonality, the total symbol length is $13.6\mu s$ including a guard interval of $0.8\mu s$ to mitigate inter symbol interference in multipath environment. An OFDMA subchannel consists of 32 subcarriers, whereby 2 subcarriers per subchannel are reserved for the transmission of pilot signals. The subchannels are directly mapped onto a contiguous fraction of the frequency channel. This is in contrast to IEEE 802.16a, where the subcarriers of a subchannel are evenly distributed over the frequency channel in a pseudo random manner to apply a kind of spreading scheme [8]. This certainly reduces the flexibility of the scheduling since the diversity of the subchannel is lost. Three different combinations of modulation schemes are used for the performance evaluation, namely QPSK 3/4, 16QAM 3/4 and 64QAM 3/4. These schemes are called PHY mode in this work. An important feature of the OFDMA PHY is the possibility of exploiting multi-user diversity. The general MAC structure is based on a centrally controlled scheme like IEEE 802.16. The frequency channel is divided into MAC frames. In this protocol, the frame length is calculated using the symbol length and the number of slots per frame. Here it is considered that the symbol length or duration is $13.6\mu s$ and symbols per slot are 8. Slots could be taken as any number greater than 2 because one slot is reserved for the signaling overhead. The signaling overhead includes the transmission of control data packets. These packets comprise messages for channel estimation and resource allocation. The remaining slots are scheduled for downlink transmissions as uplink was beyond the scope of this work. Overall, a fixed signaling overhead of one time slot is assumed independent of the amount of used resources.

6 Simulation Results

The simulation of this work is performed extending NS2 [9] whereby the statistical evaluation is conducted following [10]. Here for the traffic source model, a two-state Markov Modulated Poisson Process (MMPP) [4] is used. The two-state MMPP is in either ON state or is in OFF state. In this model packets are only generated in the ON state with fixed arrival rate α. The time spent in ON and OFF states is exponentially distributed with mean α^{-1} and β^{-1} respectively. It is also called Interrupted Poisson Process (IPP) with fixed inter-arrival time. The position of the MTs is updated at each 1ms whereby the displacement of the MTs is characterized by the Brownian motion mobility model [1]. In this model MTs move in the adjacent segment with reflecting the mobility region and the

position of the mobile user is always changed by 5m. The simulation results are presented in the subsequent subsections.

6.1 Balancing Resource Utilization

Before going to the performance analysis of newly developed DCA scheme, the correctness of the algorithm must be proved. In this regard, we have to show whether the algorithm is reacting on the various traffic situation or not. Reacting on various traffic load means that the APC will assign more subchannels in case of high traffic load situation than the case of low traffic load situation. In order to prove it, we have simulated a three cell circular scenario where each cell has 26 MTs and the radius of the mobility region is 300m. In the scenario, (Fig. 3) each AP has been assigned different traffic loads per downlink connections. The APC updates resources at the frame interval 10 while it updates the mutual SIR value at the frame interval 20. AP_0 is assigned load 3000kbps, AP_1 is 500kbps and AP_2 is 10000kbps. In this scenario, for the traffic load the mean on time is 0.1 and mean off time is 0.4, OFDMA subchannels are 32 and time slots are 17. There is no reuse of subchannels as the spatial distances between the

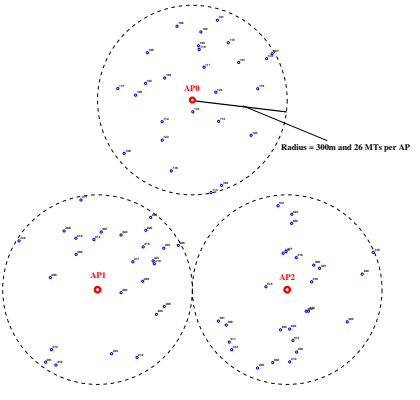
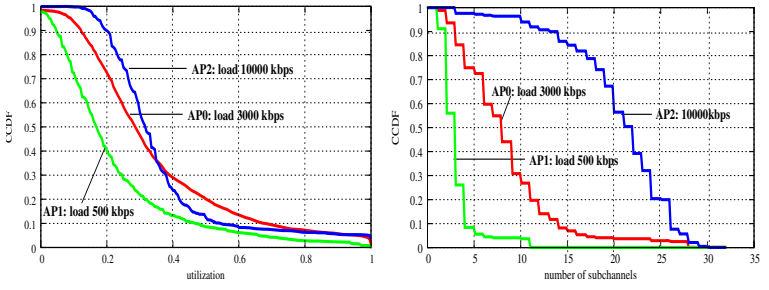


Fig. 3. Three cell scenario

APs are not sufficient for the resource reuse. The resource utilization curve is shown in Fig. 4(a). The X-axis is for the utilization that ranges from $0-1$ and Y-axis is for the Complementary Cumulative Distribution Function (CCDF) of the corresponding utilization. The CCDF decreases with the increasing assigned utilization. In the curve on average AP_0 is utilizing 35% of the assigned resources, AP_1 is 18% and AP_2 is 37%. These average values indicate that the APs are utilizing resources according to the assigned traffic loads. The APC has balanced the resource utilization of all the APs by assigning subchannels according to the traffic load. This effect can be seen in Fig. 4(b). In the curve the subchannels assignment to each AP with the corresponding traffic load is shown. As AP_2 has highest resource requirement, it has got the highest number of subchannels

whereas as AP_1 has lowest resource requirement, it has obtained lowest number of subchannels. In case of AP_0, the number of subchannel assignment is in the middle position. It has obtained a higher number of subchannels than that of AP_1 and a lower number of subchannels than that of AP_2. So from analyzing the number of obtained subchannels and utilization with various traffic loads, we can say that the developed algorithm in this paper works correctly.



(a) Utilization (b) Subchannels

Fig. 4. Resource utilization and subchannel assignment on various traffic load

6.2 Finding Optimal FRP for a Five Cell Scenario

In order to compare the newly developed DCA scheme with the fixed FCA scheme, we have investigated the optimal FRP for the scenario shown in Fig. 5. In this case, we have used 16 OFDMA subchannels, 10 MTs per cell and 9 time slots per subchannel and both mean on and mean off time are 0.1 in traffic load generation. We have obtained the optimal FRP for this scenario by investigating three feasible FRPs that are as follows:

$cosetA = \{AP_0, AP_1, AP_3, AP_4\};\{AP_2\},$
$cosetB = \{AP_0, AP_3\};\{AP_2\};\{AP_1, AP_4\},$
$cosetC = \{AP_0\};\{AP_1\};\{AP_2\};\{AP_3\};\{AP_4\}$

In *cosetA*, the average number of subchannels assigned to each set is 8, in *cosetB* is 5.33 and in *cosetC* is 3.2. Here different simulations have been performed by varying the loads of AP_2 and AP_4 while keeping fixed the loads of AP_0, AP_1 and AP_3. In the corresponding simulation result curves, *cosetA* will be termed as 2-co-channel cells because it has two clusters. Similarly, *cosetB* will be 3-co-channel cells sets and *cosetC* will be 5-co-channel cells sets. Finally, by simulating the same scenario with these three reuse partitioning we found that *cosetB* is the optimal reuse partitioning in terms of mean delay. The simulation results show that the mean delay of *cosetB* is less than that of *cosetA* and *cosetC* shown in Fig. 6(a) as in *cosetA*, the co-channel interference increases by allowing AP_0, AP_1, AP_3 and AP_4 to reuse their subchannels; therefore, the Packet Error Rate (PER) increases (Fig. 6(b)) which shows that the PER for the *cosetA* is greater than the maximum allowable PER 0.01 in the considered system. For *cosetC*,

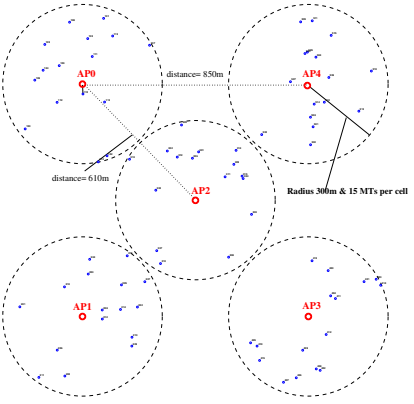
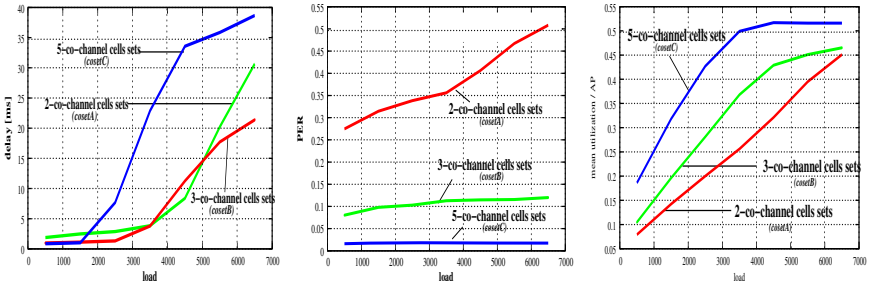


Fig. 5. Five cell scenario

the mean PER is less than 0.01 since each cell uses orthogonal subchannels, and consequently, there is less co-channel interference.

Although in case of *cosetB* mean PER is higher (there is reuse of subchannels) than *cosetC*, mean delay decreases on the higher traffic loads for the better utilization of the resources than the *cosetC*. In case of *cosetC*, delay increases on the higher traffic loads due to the higher resource requirement of AP_2 and AP_4. As they do not get more subchannels on increasing load, the packets are queued, and therefore, the mean delay increases. The mean utilization curve is shown in Fig. 6(c). The mean utilization is around 50% because different simulations



(a) Mean delay (b) Mean PER (c) Mean utilization

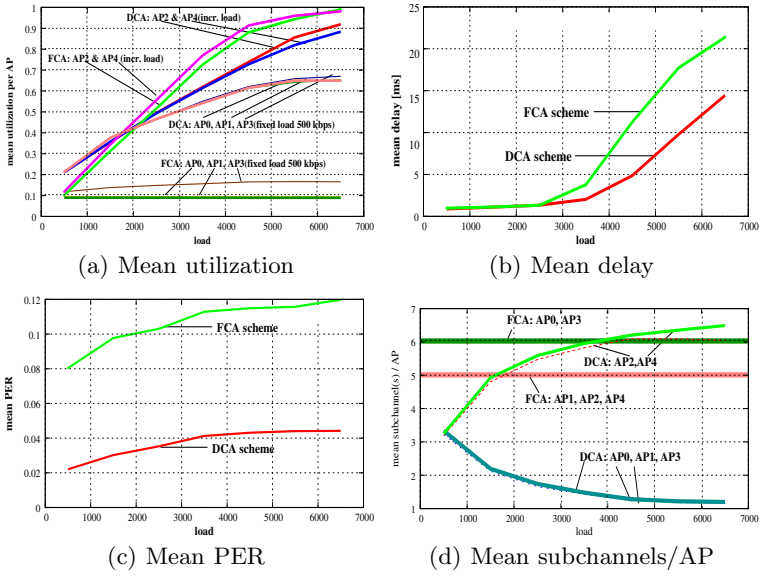
Fig. 6. Comparison of mean utilization, PER and delay among 3 FRPs

have been performed only increasing the load for two APs (AP_2 and AP_4) by keeping others fixed. Here the utilization of the APs whose loads were fixed is too low and thereby, the mean utilization decreases to 50%. The curve shows that the utilization for *cosetC* is the highest, and with the increasing traffic load it becomes saturated due to the limited queue length. But in case of *cosetA*, the

mean utilization is upward at the high load as in this case packets retransmission is higher for the co-channel interference.

6.3 Comparison Between Developed DCA Scheme and FCA Scheme

In this subsection, we compare the performance of the newly developed Dynamic subChannel Allocation (DCA) scheme with Fixed subChannel Allocation scheme (FCA) using the same scenario (Fig. 5) discussed in the previous subsection. Here for the FCA scheme the optimal cell FRP *cosetB* (discussed in the above subsection) has been used. In this partitioning, co-channel set $\{AP_0, AP_3\}$ gets 6 subchannels while the remaining two co-channel cells sets, $\{AP_1, AP_4\}$ and $\{AP_2\}$ get 5 subchannels at each. As DCA scheme always performs better than FCA scheme in an uneven traffic situation, we have done different simulations with increasing traffic load for AP_2 and AP_4 while keeping fixed the traffic loads of AP_0, AP_1 and AP_3 at 500kbps. As a result, different resource requirement behaviors were prevailed in the APs. The mean utilization of resources for each of the APs is shown in Fig. 7(a). In case of the FCA, in the mean utilization curves, AP_0, AP_1 and AP_3 have always a lower utilization as their traffic load is fixed at 500kbps. Their average utilization is around 10%. On the other hand, AP_2 and AP_4 are over loaded in case of higher traffic load. Their utilization is greater than 80% after the load 3500kbps. Whereas in the DCA, AP_0, AP_1



(a) Mean utilization (b) Mean delay

(c) Mean PER (d) Mean subchannels/AP

Fig. 7. Mean delay comparison between DCA and FCA schemes on increasing load of AP_2 and AP_4 and fixed load of AP_0, AP_1 and AP_3

and AP_3 utilize 50% of the assigned resources on average and AP_2 and AP_4 utilize 70% of the assigned resources on average. The mean delay for the FCA scheme is greater than that of the DCA scheme (shown in Fig. 7(b)) as the FCA scheme is not assigning sufficient subchannels to AP_2 and AP_4 on increasing load. Another reason is that the PER increases as the FRP of cells is used by the FCA scheme which is shown in Fig. 7(c). But in case of the DCA scheme, as adaptive cell reuse partitioning is used, the mean PER is much lower than the FCA. The mean number of assigned subchannels for each the AP is shown in Fig. 7(d). As in the used scenario the APs are not sufficiently far apart to reuse resources and the loads of AP_2 and AP_4 are varying, both are getting higher number of subchannels on increasing load. On the other hand, as the loads of the remaining three APs (AP_0, AP_1 & AP_3) are fixed, they are getting lower number of subchannels with increasing loads of AP_2 and AP_4.

7 Conclusion and Future Work

This work shows that the intelligently handled interference aware DCA scheme has a great benefit compare to FCA schemes in case of uneven traffic situation. This is due to the fact that in the uneven resource requirements, the DCA scheme attains more utilization of resources by assigning resources to the APs that actually need resources. In general, there is a trade off between the QoS, the implementation complexity of the channel allocation algorithms, and the spectrum utilization efficiency [6]. Therefore, the FCA scheme becomes superior at high offered traffic load, especially in case of uniform traffic situation and DCA scheme performs better in case of uneven traffic loads. Most of the previously developed DCA schemes have involved PER and SINR for taking the channel allocation decision. As a result the computational complexities of those algorithms were either exponential or too high. Whereas by considering the utilization as the decision factor, the computational complexity of the developed algorithm has been reduced to $O(n^2 \cdot m) + O(k \cdot n^2)$ and it works better than the FCA scheme with the optimal fixed reuse partitioning.

The further improvements of this algorithm can be done by prioritizing traffic. In this regard, the APs should provide individual information of utilization of resources both for high and low priority traffics. After gaining the full knowledge about the traffic priorities of the individual APs, the APC will assign subchannels according to that knowledge. Another improvement can be done by using adaptive reuse threshold for the decision concerning the reuse of resources. In this regard, the APC can trace the modulation schemes that the APs use in data transmission and can set the reuse threshold according to the modulation schemes or PHY mode.

Acknowledgement

We would like to express our sincere gratitude to Prof. Bernhard Walke for the fruitful discussions towards the success of this work. This project has been

jointly funded by Siemens AG, Munich, Germany and Chair of Communication Networks, RWTH Aachen University, Germany.

References

1. Bettstetter, C: Mobility modeling in wireless networks: categorization, smooth movement, and border effects. ACM SIGMOBILE Mobile Computing and Communications Review. **5** (2001), 55–66
2. Cheong, S., Cheng, W., Letaief K.: Multiuser OFDM with Adaptive Subcarrier, Bit, and Power Allocation. IEEE Journal on Selected Areas in Communications. **7** (1999), 1747–1758
3. Choi, M., Hanzo, B.J., Munster L., Keller, T.: OFDM and MC-CDMA for Broadband Multi-User Communications, WLANs and Broadcasting. Wiley, United Kingdom. (2003)
4. Fischer, W., Meier-Hellstern, V.: The Markov-modulated Poisson process (MMPP) cookbook. Performance Evaluation. **18** (1993), 149–171
5. Furukawa, H., Akaiwa, Y.: Self Organized Reuse Partitioning, Dynamic Channel Assignment Method in Cellular systems. Proceedings of the 43rd IEEE VTC. (1993), 524-527
6. Katzela, I., Naghshineh M.: Channel Assignment Schemes for Cellular Mobile Telecommunication Systems. IEEE Personal Communications, **3** (1996), 10–31
7. Kim, I., Lee, H.: On the use of Linear Programming for Dynamic Subchannel and bit Allocation in Multiuser OFDM. IEEE Global Telecommunica tions Conference. **6** (2001), 3648–3652
8. Koffman, I., Roman, V.: Broadband Wireless Access Solutions based on OFDM Access in IEEE 802.16. IEEE Communication Magazine. **40** (2002), 96–103
9. NS-2: http://www.isi.edu/nsnam/ns/NS-2
10. Org, C.G., Schreiber,F.: The RESTART/LRE Method for Rare Event Simulation. In Proceedings of the Winter Simulation Conference. (1996), 390–400
11. Pietrzyk S., Janssen G.: Multiuser Subcarrier Allocation for QoS Provision in OFDMA Systems. Proc. VTC. **2** (2002), 1077–1081
12. Tuerke, U.: Centralized Dynamic Channel Allocation in HiperLAN/2. Diploma thesis, ComNets RWTH Aachen University (September 2000)
13. Roy, B.: Dynamic Subchannel Allocation in a Multi-cellular OFDMA System based on Interference Measurement and Traffic Situation. Master Thesis, Comnets, RWTH Aachen University (December 2005)
14. Yin, H., Liu, H.: An Afficient Multiuser Loading Algorithm for OFDM-based Broadband Wireless Systems. IEEE Global Telecommunications Conference. **1** (2000), 103–107

Sized-Based Replacement-k Replacement Policy in Data Grid Environments

HongJin Park[1] and ChangHoon Lee[2]

[1] School of Computer Information and Communication Engineering, Sangji University, Woosan-dong, Wonju-si, Kangwon-do, 220-702, Korea
hjpak1@sangji.ac.kr
[2] Department of Computer Engineering, Hankyong National University, 67, Seokjeong-dong, Ansung-si, Kyunggi-do, 456-749, Korea
be4u@hknu.ac.kr

Abstract. The data grid computing provides geographically distributed storage resources to solve computational problems with large-scale data. Unlike cache replacement policies in virtual memory or web-caching replacement, an optimal file replacement policy for data grids is the one of the important problems by the fact that file size is very large. The traditional file replacement policies such as LRU(Least Recently Used), LCB-K(Least Cost Beneficial based on K), EBR(Economic-based cache replacement), LVCT(Least Value-based on Caching Time) must predict near future or need additional resources for file replacement. In this paper, the SBR-k(Sized-based replacement-k) policy for solving previous problems propose. The SBR-k replacement is a file size based replacement policy for new file. The results of the simulation show that the proposed policy performs better than traditional policies.

1 Introduction

As an alternative to high-priced super computing, low-priced and high-efficiency grid computing is being proposed and used. Grid computing means environment that can utilize distributed high-performance computing resources through networking them regardless of organization and area. In general, grid computing is divided into computer grid, data grid and access grid. Computer grid shares geographically distributed computing power as if using one high-performance computer and provides collaborating environment so that distributed researchers can execute a project jointly. In grid computing, data grid means a platform network, on which heterogeneous high-performance nodes and data storage resources are geographically distributed. Data grid computing is applied in various areas related to generating, storing and processing distributed high-capacity data like geological research, high-energy physics, aerophysics and climate change modeling [1-4], and this thesis deals with the element technologies of data grid.

In data grid, each storage node is a high-capacity storage device, a third storage device, high-performance storage system (HPSS), etc., enabling users to create, delete, read and write data locally or remotely. In addition, critical problems in data grid

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 353 – 361, 2006.
© Springer-Verlag Berlin Heidelberg 2006

are long delay time in the interconnected network and access to the large volume of data in the storage device. To solve these problems, overhead resulting from remote access is avoided and remote data is cached locally to improve the reliability of resources. Caching technology has been used to improve the performance and reliability of storage systems such as computer system, database and Web caching. In several aspects, however, caching technology used in data grid is different from existing caching technology. For example, a network used in data grid is based on wide area network (WAN) in which the distance between nodes is long, file size is several gigabytes, and cache size ranges from hundreds of gigabytes to tens of terabytes. While existing caching technology used in the Web has a delay time of several seconds or several minutes and is optional, in data grid environment delay time in data transmission is as long as several minutes or even several hours, and the use of Web technology is also compulsory.

In data grid environment, the number of requests to a high-capacity data storage device can be tens, hundreds or thousands. In general, each request is queued and it is decide first which file will be searched first. This decision is called file admission policy. Following file admission policy, file replacement policy is executed. File replacement policy is to select a file in the storage to secure a space in the requested file. In data grid environment, the optimal file replacement strategy technique is the existing virtual memory paging replacement strategy but, unlike Web caching strategy, it involves complicated problems due to differences in file size and access pattern [5.6]. Existing file replacement policies like algorithms such as LRU and LFU select files with too simple information to be used in data grid, and file replacement strategies for data grid such as LCB-K proposed in [5] and EBR in [7] have the problem that they have to forecast requests to replace files. Moreover, the LVCT policy proposed in [8] has to maintain and manage file stacks for file replacement.

SRB-k (size-based replacement-k), the file replacement policy proposed in this study, is advantageous in that it does not need additional resources like stacks and, as it replaces files without forecasting future requests, it selects only a minimum number of files to be replaced in the future.

The structure of this paper is as follows. Chapter 2 describes a system model in data grid. Chapter 3 explains existing file replacement policies and their problems in connection to the policy proposed in this study. Chapter 4 explains the policy proposed in this study. Chapter 5 compares the performance of the proposed policy with that of existing ones and draws conclusions.

2 Model of the Data Grid System

In data grid environment, a storage resource manager is a middleware component essential for sharing data and storing and managing resources. A major function of a storage resource manager is caching high-capacity disk information. A storage resource manager can be specialized into a disk resource manager or a hierarchical resource manager (HRM) [6].

Fig. 1 shows storage resource managers specialized in data grid environment presented in this study. Each site can have one or more storage resource managers and they are interlinked through WAN. In addition, resources with each site are organized

through local area network (LAN). In each site are files of various sizes, and a specific file can be copied to multiple sites in the data grid.

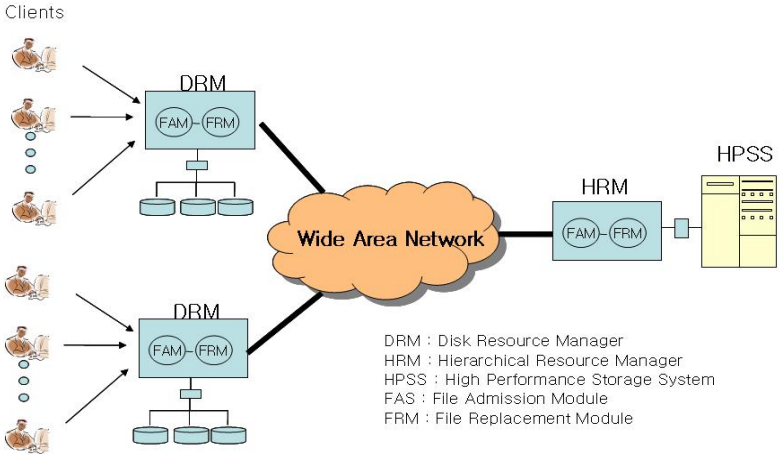


Fig. 1. Storage Resource Manager in Data grid

A disk resource manager maintains a metadata information table on files (e.g. file name, size, location) and, by a user's request for a resource, it provides the corresponding resource. First, it checks the metadata information table to see if the resource requested by the user exists locally, and if it is the manager provides the resource immediately and if not it requests the resource from a remote place and copies it.

In case a new file is created in a site, the resource storage manager in the site sends the meta information of the file (file name, size, etc.) to all the other sites and, later, sends the file if requested. In addition, if an original file is copied from another site, its information (meta information) is sent to other sites. Sites that have received meta information can calculate the cost of processing the corresponding file through <Equation 1> below. In <Equation 1>, v1 and v2 indicate sites, and it shows the cost of processing file F from site v1 to v2.

$$Cost(f, v1, v2) = latency + file\_size(fi) / bandwidth(v1, v2) \quad \text{--------} \quad \text{<Equation 1>}$$

Based on <Equation 1>, the resource storage manager in each site updates information on the file in its own file meta information table.

The hierarchical resource manager manages high-performance storage devices.

Different from Web proxy cache, the number of requests arriving simultaneously at a storage resource manager can be hundreds or thousands. Many users are connected to data grid and support resource requests. In response to this, the storage resource manager queues requested resources in the request buffer. For queued resources, the manager decides which file should be processed first (e.g. FIFO) and the decision is called "file admission policy." In (Figure 1), the file admission policy is executed by the file admission module (FAM) inside the storage resource manager. To process an

admitted file request, it performs a local search and, if the file is not found, has to get the requested file from a remote site. At that time, if the local disk space is not sufficient, the manger has to select victims to be replaced among the stored files to create space for the requested file and this is called "file replacement policy." This task is performed by the file replacement module (FRM) in the storage resource manager. This study is about the file replacement policy within a storage resource manager in data grid environment.

3 Relevant Researches

The most well-known page replacement policies used commonly in operating systems are LRU (Least Recently Used) and LFU (Least Frequently Used) [9], and LRU-k [10] is the mixture of LRU and LFU, applying LRU based on k reference in the past. These policies are algorithms that select files to be cached based on the time of the latest reference (LRU) or the frequency of reference (LFU).

LCB-K (Least Cost Beneficial based on K backward reference) proposed in [5] is a policy for file replacement in data grid. The algorithm uses a utility function to select files. The utility function based on LRU-k policy makes calculations, forecasting the rate of files to arrive in the future. To forecast the future arrival rate, it uses '(the number of recent references up to K * the cost of search) / file size.' Each file to be cached is calculated using the utility function and its relative order is assigned, and file that has the lowest utility value is selected first and replaced. The algorithm continues to select files with low utility value and replace them in order to secure space for requested files.

EBR (Economic-Based cache Replacement) policy was proposed in [7]. The policy optimizes file redundancy in data grid environment. It uses an economical model that replaces the least cost for redundant files in order to decide the optimal redundancy. If there is enough space, a newly arrived file is stored into the disk automatically. If the disk does not have enough space, EBR selects a file with the least value in the disk. For the selection, it uses a replica optimizer implemented in each site. To maximize the profit, the replica optimizer logs the value of each file in the storage to and uses it as an input to the future income forecast function. The forecast function calculates a value by estimating requests at time W in the future based on requests at window time W in the past considering temporal relevancy, geographical relevancy and sequential relevancy.

LVCT (Least Value-based on Caching Time) policy was proposed in [8]. The policy replaces files considering how soon the files will be re-accessed (caching time). The file(s) with the lowest utility value are selected, and the utility function is (1/caching time)* cost/file size. The policy uses a caching time stack that contains caching time and file size. That is, caching time, which is set based on time for accessing each file, and file size are stored in the caching time stack. For example, if file F is accessed to be replaced, it is moved to the top of the stack and its caching time is reset at 0. Again, a re-accessed file is moved to the top of the stack. In this way, on the bottom of the stack are the files that have been least accessed and they are replaced. That is, files with long caching time are least accessed ones, and they are replaced.

Existing policies presented above have the following problems. LRU, LFU and LRU-k are well-known policies but they select files based on too little (or too simple) information. The major problems of LCB-K policy proposed in [5] are that the utility function forecasts the future arrival rate of each file and that files selected to be replaced have to have been accessed at least once within a certain length of time (k) in the past. Accordingly, the utility function is not applied to files outside the time, so they cannot be candidates for replacement. For this reason, the file selected by the policy cannot be the optimal choice for replacement. The problem of the policy proposed in [7] is that, for economical modeling, it has to forecast future requests based on past information. LVCT policy has to maintain a stack containing caching time and file size. In addition, it requires an additional work of moving accessed files to the top of the stack. As a whole, future situation should be forecasted to secure space, and it is likely to replace all files smaller than the requested file to obtain space in need. Moreover, unnecessary resources like stack have to be maintained for replacement.

4 Proposed Algorithm

The algorithm proposed in this study is SRB-k (Size-Based Replacement - k). The proposed algorithm considers file size to reduce the number of files corresponding to a requested file rather than forecasting the uncertain future for replacement.

In data grid environment, hundreds or thousands of file requests can arrive simultaneously at the local storage resource manager, and the storage resource manager queues them first. Queued request messages are sent to the file replacement module (FRM) in the storage resource manager after the module decides which file should be searched first.

The file replacement module checks if the requested file is in the local disk. If it is, the file is delivered to the user who requested the file. If not, the file is requested to a remote disk and is cached in the local disk. At that time, if there is no space for the new file in the local disk, files to be replaced are selected by the file replacement algorithm. All files in the local storage are the candidates for replacement and they are called “unpinned files,” and those selected for replacement from the unpinned files are called “pinned files.”

The file replacement algorithm proposed in this study is as in [Table 1]. If r is the size of a file requested by a user, the SRB-k algorithm first finds a file, the size of which is r, in the local storage. If there is a file of size r in the storage, the file is selected for replacement and this is the optimal replacement in the SRB-k algorithm.

If there is no file of that size, SRB-k executes the file replacement algorithm considering the k value, which is proportional to the size of the file requested by the user. For example, if the size of the requested file is 1000MB and k is 0.1 (or 10%), the size of k is 100MB, namely, 10% of the size of the requested file. Assume that there is no file of that size in the local file, k is 0.1, q is the sum of the size of the requested file and the k value (1100MB), and p is file(s) in the local disk, the size of which is larger than r and smaller than q. Then the SRB-k algorithm selects a file to be replaced among p file(s) considering LRU, and replaces it. If there is no p in the local disk, the

SRB-k algorithm looks for files, the size of which is 900MB (q'), the size of the requested file with the deduction of the k value. The SRB-k algorithm finds file(S), the size of which is smaller than r and larger than q'. The SRB-k algorithm selects a file to be replaced among the p' file(s) and replaces it. If there is no p' file in the local disk, the SRB-k algorithm is repeated with increasing the k value until a file to be replaced is found.

Table 1. Proposed SRB-k replacement policy

```
if(new_requested_file_size == unpinned_file_size)
        then {
            replace file;
            exit;
        }
   k=0;
   while( 1 ){
       k += 0.1
    k_size = new_requested_file_size * k
   i) search files with following condition
              (new_requested_file_size < files < (new_ requested_file_size + k_size))
            select file considering LRU algoruthm
            replace file;
            exit;
    ii)  search files with following condition
                ((new_requested_file_size – k_size) < files < new_ requested_file_size)
            select file considering LRU algoruthm
            replace files;
             exit;
```

5 Evalution

The efficiency of cache replacement policies is commonly evaluated by hit ratio and byte hit ratio [5,6]. Hit ratio indicates the ratio of the number of requests for a file existing in the cache to the whole number of requests and byte hit ratio is the ratio of the volume of cached data to the volume of requested data. Hit ratio and byte hit ratio are used to measure improvement in response time and the efficiency of bandwidth.

The simulation tool used in this study is OptorSim. OptorSim [11,12] is a data grid simulation tool implemented in Java by the European DataGrid project (We set up Java 1.5.0_04 and Ant 1.6.5 to install OptorSim). Simulation environment was based on [5,6]. The range of file size was set between 1.0 and 2.1GB. For requests, we used Poisson arrival time of 10 seconds on the average. File size was distributed evenly between 5000,000 byte and 2,147,000,000 byte. Files were requested at certain intervals, and the locality of reference with the 80-20 rule was applied for reference. The 80-20 rule means that 80% of requests re-refer to 20% of files. A total of 1.1 terabyte could be stored in 20 sites (18 sites of 50 gigabyte capacity and 2 sites of 100 gigabyte capacity).

Fig. 2 and Fig. 3 show the average hit ratio and byte hit ratio of the buffer of each site for the proposed policy, LRU and RND (Random). As a whole, if the buffer size is small, there is no big difference in hit ratio among the proposed policy, LRU and RND, but hit ratio rises with the increase of buffer size. In Fig. 2, hit ratio is similar until the buffer size reaches 0.5% or 5.5 gigabyte (proposed policy 0.1, LRU 0.8, RND 0.7), but when the buffer size is 2% or 22 gigabyte, the proposed policy shows higher performance in hit ratio (0.25) than LRU (0.16) and RND (0.11). This is because the proposed policy selects the file closest in size to the new file.

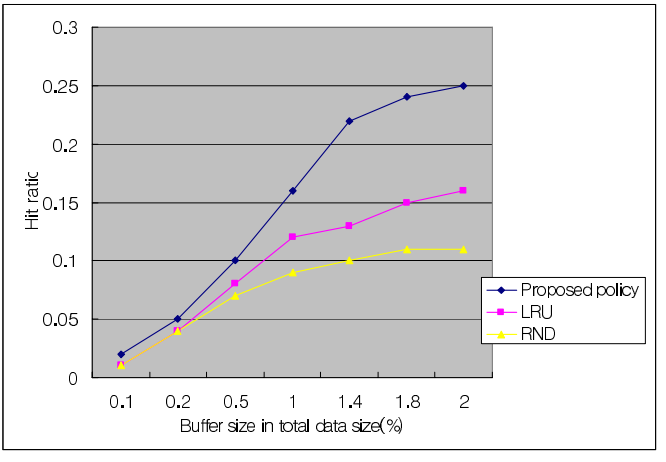


Fig. 2. Hit ratio in data grid

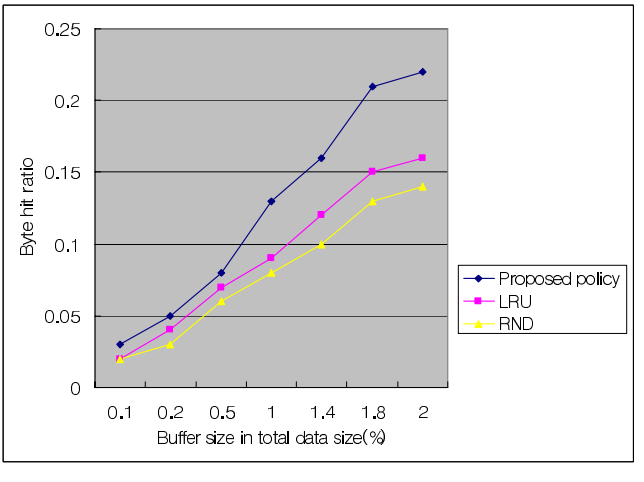


Fig. 3. Byte hit ratio in data grid

6 Conclusion

The efficiency of cache replacement policies is commonly evaluated by hit ratio and byte hit ratio [5,6]. Hit ratio indicates the ratio of the number of requests for a file existing in the cache to the whole number of requests and byte hit ratio is the ratio of the volume of cached data to the volume of requested data. Hit ratio and byte hit ratio are used to measure.

Research is being made continuously on data grid computing for efficient processing of geographically distributed high-capacity data. Different from file replacement policies in existing Web caching virtual memory, file replacement strategies in data grid environment are complicated due to large file size and the high capacity of the cache. Policies proposed in previous researches such as LRU, LCB-K, ERB and LVCT replace files based on too simple information, require additional resources, or have to forecast the uncertain future. To solve these problems, this study proposed SRB-k that replaces files based on file size. According to the results of performance evaluation, hit ratio was similar when the cache size was small, but the proposed policy was superior to LRU and RND when the cache size was large.

A task for future study is executing the proposed file replacement policy considering the priority of files. In addition, if files contain multimedia data, different policies may be applied according to the type of media in consideration of limited storage space.

References

1. B. Allcock, I. Foster, V. Nefedova, A. Chervenak, D. Deelman etc, " High-performance remote access to climate simulation data : A challenge problem for data grid technologies" Proceeding of the SuperComputing Conference, Non., 2003
2. K. IIoltman, "Data grid system overview and requirements", Technical report , CERN July 2001
3. M. Russel, G. Allen, G. Daues, I. Foster, E. Seidel, J. Novotny, J. Shalf and G. von Laszewski, " The astophysics simulation collaboratory : A science portal enabling community software deveopment", Cluster Computing , 2002
4. A. Chervenak, I. Foster, C. Kesselman, C. Salisbury and S. Tueckes, "The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets", Journal of Network and Computer Applications, pp. 187-200, 2002
5. E. J. Otoo, F. Olken and A. Shoshani, " Disk Cache replacement algorithm for storage resource managers in data grids", In The 15th Annual Super Computer Conference, pp. 1-15, Nov., 2002
6. J.H Abawajy, "File Replacement Algorithm for Storage Resource Managers in Data Grids", LNCS 3038, pp. 339-346, 2004
7. M. Carman, F. Serafini, L. Stokinger, K. Stockinger, "Towards an Economy-Based Optimisation of File Access and Replication on a Data Grid", In Procedings of 2nd CCGRID, pp. 120-126, 2002
8. S. Jiang, X. Zhang, "An Efficient Distributed Disk Caching in Data Grid Managemnet", In Proceedings of Cluster Computing, 2003

9. S. Aberham, G. Peter Baer, G. Greg, Operating system principles, 7th, Wiley, 2006
10. E. J. Otoo, F. O'Neil, G. Weilum, "The LRU-K page repalcement Algorithm for Database Disk Buffering", The 15th Annual Super Computer Conference, Nov. 2002
11. Daid G. Gameron, Ruben Carvajal-Schiaffino, Jamie Ferguson, OptorSim v2.0 Installation and User Guid, http://cern.ch/edg-wp2/optimization/opotorsim.html
12. W.H.Bell, D.G. Gameron, "Optorsim - data grid simulator for studying dynamic data replication strategies", Journal of High Performance Computing Application, 2003

Barrier Elimination Based on Access Dependency Analysis for OpenMP

Naoki Yonezawa[1], Koichi Wada[2], and Takahiro Aida[2]

[1] Kanagawa University, Hiratsuka Kanagawa 259-1293, Japan
yonezawa@info.kanagawa-u.ac.jp
[2] University of Tsukuba, Tsukuba Ibaraki 305-8573, Japan
wada@cs.tsukuba.ac.jp, t_aida@padc.cs.tsukuba.ac.jp

Abstract. In this paper, we propose a new compiler technique for eliminating barrier synchronizations. In our approach, the compiler collects access information about array accesses and analyzes data dependency. If there was no dependency, barrier synchronizations can be eliminated. Additionally, even if the dependency was detected, there are cases when the barrier synchronization can be replaced with send-receive pairs of communications. For evaluation, we executed two application programs: Jacobi Method and Gaussian Elimination, on a PC cluster with barrier elimination applied. For comparison, we also executed the programs before elimination of barrier synchronizations. With barrier elimination, 1) the execution time is always reduced, and 2) as the number of processors increases, the reduction ratio of the execution time also increases. For 16 processors, we obtained 19.00% and 50.36% of the reduction ratio for Jacobi Method and Gaussian Elimination respectively.

1 Introduction

Recently, OpenMP [1] attracts attention as a language for shared memory environment since programmers can parallelize the existing sequential programs by appending directives. However, large-scale machines which have physically shared memory are not popular compared with PC clusters. Many researchers have developed some underlying mechanism so that virtually shared memory is realized on distributed memory environment [2,3,4]. We have developed the compiler called *Quaver* [5], which enables OpenMP programs to be executed on distributed memory environment. Quaver identifies the segments in the source code that access the shared data and finds the producer-consumer relationship. Then, Quaver generates communication code, in which a producer sends appropriate data to the consumer using message passing library.

In distributed memory environment, since the cost of communication via network is very expensive, many techniques for reducing or hiding communication overhead have been proposed so far; such as data caching, optimizing data assignment, relaxing memory consistency model, and overlapping communication with computation.

In this study, we focus on the cost of barrier synchronization. A barrier synchronization is inserted in a program in order to guarantee that an event above

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 362–373, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the barrier synchronization completes by the point of leaving barrier synchronization and all processors which execute the program in parallel see the same result of the event. Because barrier synchronization is implemented as the transferring all-to-one and one-to-all messages in distributed memory environment, the cost of barrier synchronization grows quickly as the number of processors increases. Additionally, even if the loads among the processors are completely balanced, barrier synchronization yields idle time of the processors because of the randomness caused by external network delays and scheduling algorithm which an operating system adopts.

In this paper, we propose a new compiler algorithm which eliminates barrier synchronizations. As mentioned above, Quaver generates send-receive pair in the program. Our idea is based on the fact that the pair of communications guarantees the order of the events, i.e., the order of the accesses to virtually shared memory. In our approach, if data dependency can be analyzed at compile time or statically, a barrier synchronizations is replaced with several send-receive pairs. As opposed to Tseng's algorithm [6], which eliminates barrier synchronization if there is no data dependency between the consecutive phases divided by the barrier synchronization, our approach can be applied even if the data dependency exists.

The rest of this paper is organized as follows. At first, we describe our OpenMP compiler, called Quaver, in Sect. 2. An array section descriptor, which is a component of Quaver, is also described in this section. Section 3 proposes a new algorithm for eliminating barrier synchronizations. Section 4 presents the results of performance evaluations. Finally, we conclude in Sect. 6.

2 *Quaver*: An OpenMP Compiler Based on Array Section Descriptor

2.1 Array Section Descriptor for Parallel Computing

An array section assigned to a processor in block-cyclic manner is mainly accessed by the processor on which the array section is placed. Those accessed sections consist of iteratively accessed subsections that include non-accessed subsections, as shown in Fig. 1(a). In the case that the processor accesses the slightly wider section than the assigned section, the shape will be similar to the above case (Fig. 1(b), where the increased length is denoted by α). Our proposed quad [7] can represent these access patterns concisely, which typically occur in parallel programs.

Quad, as its name implies, is a quadruplet of integers. A quad (a, b, c, d) consists of the following parameters:

a: first subsection's offset from the head of the array,
b: the length of accessed subsection,
c: the length of non-accessed subsection,
d: the number of repetitions.

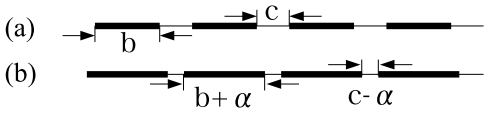


Fig. 1. Typical Access Patterns Observed in Parallel Programs

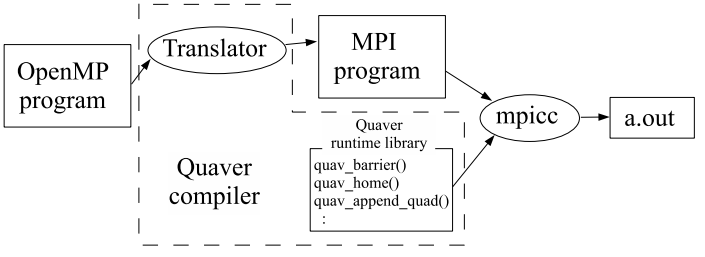


Fig. 2. The Flow of Compiling

We have defined the intersection operation and the union operation among quads. The intersection operation is used to investigate data dependency and to identify data to be transferred from a producer to consumers. The union operation is used to merge several quads into fewer quads.

2.2 Compiling an OpenMP Program by Quaver

Quaver consists of code translator and runtime library, as shown in Fig. 2. The code translator analyzes an OpenMP program and locates the shared data accesses, which are represented by several quads. If Quaver finds an OpenMP directive, it inserts the appropriate library functions defined in Quaver runtime library. The final code generated by Quaver is compiled by `mpicc` and can be executed on a PC cluster. At runtime, data to be transferred will be identified by applying the intersection operation among quads at the point of synchronization and sent by a message passing call such as MPI_Send().

In the rest of this section, we describe Quaver runtime library and how Quaver analyzes an OpenMP code and inserts the library functions to the code.

2.3 Quaver Runtime Library

Quaver runtime library consists of the following functions.

quav_append_quad() registers access information including quad which represents the accessed array section, the access type (i.e., read or write), and the processor ID to the *quad table*. The quad table is indexed by access type and processor ID.

quav_divide_loop_and_append_quad() provides a facilitative method which is equivalent to 1) computing array section accessed by a processor in a loop which is directed to divide its iterations for parallel execution, 2) generating quads which represent the divided section, and then 3) calling quav_append_quad().

quav_assign_home() is called after a shared array has been declared. It divides the array equally and assigns the divided subarray to each processor, that is *home processor* of the subarray. It is guaranteed that a home processor has latest data after barrier synchronization.

quav_home() is typically called in the body of parallelized loop. It receives processor ID and loop index value as parameters and returns true if an iteration should be executed by the processor. The parallelized loop is executed mutual-exclusively by using quav_home().

quav_barrier() blocks the execution of a processor until all the processors have arrived at the barrier synchronization. Data to be read by consumers after this barrier synchronization are identified and pushed to consumers in quav_barrier().

quav_fetch() dynamically fetches data from its home processor at runtime if data to be read cannot be identified at compile time. Using this function, any program can be executed even if it is difficult or impossible to analyze accesses to the data because of dynamic behavior of the program. If the data can be identified at compile time, Quaver does not generate quav_fetch() because it is guaranteed that the data arrive at consumer's memory by the previous quav_barrier() returns.

2.4 Generating Communication Code

In analyzing an OpenMP program, Quaver locates the shared data accesses. If the access is write, it is represented by quads and quav_append_quad() is generated in the location on which the write access occurs. If the access is read, quav_append_quad() is generated in the location on which the corresponding write access occurs, which produces data read by the read access.

At the point of synchronization, the code translator inserts quav_barrier(). To identify data to be transferred to the consumer, quav_barrier() picks up two quad sequences from the quad table. The first sequence represents the array section written by the processor that is executing the quav_barrier(). The second one represents the array section read by one of other processors. Then, quav_barrier() invokes intersection operation between these two quad sequences and consequently a new quad sequence representing data to be transferred are obtained. If the quad sequence is not empty, the data represented by the sequence are transferred to the consumer.

Figure 4 depicts an example of code segment which is generated as a result of translating the original OpenMP program shown in Fig. 3. In Fig. 4, inserted codes are emphasized in italics.

```
#pragma omp parallel for
for (i = 0; i < 16; i++)
   a[i] = i * i;
b = a[2] + a[3] + a[4] + a[5];
```

Fig. 3. An OpenMP Program

```
quav_divide_loop_and_append_quad(a, write, mypid, a(0, 16, 0, 1));
#pragma omp parallel for
for (i = 0; i < 16; i++) {
   if (! quav_home(a, i, mypid)) continue;
   a[i] = i * i;
}
quav_append_quad(p0, read, a(2, 4, 0, 1));
quav_barrier();
if (mypid == p0)
   b = a[2] + a[3] + a[4] + a[5];
```

Fig. 4. Inserting Quaver Runtime Library

3 A New Technique for Eliminating Barrier Synchronizations

3.1 Library Functions to Replace with Barrier Synchronizations

In order for Quaver to eliminate barrier synchronizations, we append two new functions to the runtime library.

quav_append_quad_to_recv_data() is similar to quav_append_quad() but it appends a quad to the different entry of the quad table. Typically, the quad to be appended represents read data which a process calling this function accesses at the next phase and is used to compute the amount of data to be received in quav_send_recv_if_needed().

quav_send_recv_if_needed() calculates the amount of data to be received and receives the data. Additionally, data to be read by consumers are identified and pushed to consumers similarly in quav_barrier(). If all of the data to be received arrives, the function returns.

3.2 A Compiler Algorithm to Eliminate Barrier Synchronizations

Figure 5 shows our proposed algorithm for eliminating barrier synchronizations.

If read accesses which will occur at the next phase are identified at the current phase, the data can be transferred to consumers in advance. If *all* read accesses which will occur at the next phase are identified at the current phase, transferring the data to consumers can replace with the barrier synchronization which divides the consecutive phases.

```
void eliminate_barrier(void)
{
    foreach (codes) {
        tree_node.all_hosited = analyze_basic_block();
        if (found an implicit barrier) replace it with quav_barrier();
    }
    foreach (tree_nodes) {
        if (found a quav_barrier() and tree_node.all_hosited) {
            generate quav_append_quad_to_recv_data();
            replace quav_barrier() with quav_send_recv_if_needed();
        }
    }
}

int analyze_basic_block(void)
{
    int all_hoisted = 1;
    if (found a basic block) all_hoisted &= analyze_basic_block();
    if (failed to hoist read accesses) all_hoisted = 0;
    return all_hoisted;
}
```

Fig. 5. An algorithm for eliminating barrier synchronizations

In analyzing a basic block of OpenMP program, Quaver tries to hoist read accesses. In hoisting, if all read accesses in the block are hoisted to its upper basic block, Quaver sets a flag called *all_hoisted* to true and propagates it to the upper basic block. When Quaver reaches the barrier synchronization, if all_hoisted flag is true, it shows that all read accesses are collected. In this case, quads which represent the read accesses are appended to the quad table by quav_append_quad_to_recv_data() and quav_send_recv_if_needed() replaces with quav_barrier().

Read accesses can be hoisted even if the block contains a call to a procedure. In analyzing the procedure, if quads on all read accesses are represented using its parameters and/or global variables and are hoisted to the beginning point of the procedure, all_hoisted flag is set. On the caller's side, if the flag is true, all variables which appear in the quads are replaced with the corresponding arguments. The resulting quads are treated in the same way as other quads.

3.3 An Example of Eliminating Barrier Synchronizations

Figure 6 shows an example of barrier elimination which is obtained by applying the algorithm to the code of Fig. 4. Note that quav_append_quad_to_recv_data() is called by only Processor 0 so that it receives fresh data.

```
quav_divide_and_append_quad(a, write, mypid, a(0, 16, 0, 1));
#pragma omp parallel for
for (i = 0; i <16; i++) {
   if (! quav_home(a, i, mypid)) continue;
   a[i] = i * i;
}
quav_append_quad(p0, read, a(2, 4, 0, 1));
if (mypid == p0) quav_append_quad_to_recv_data(p0, read, a(2, 4, 0, 1));
quav_send_recv_if_needed(mypid);
if (mypid == p0)
   b = a[2] + a[3] + a[4] + a[5];
```

Fig. 6. Eliminating Barrier Synchronization

4 Evaluation

For evaluation, we executed two application programs: Jacobi Method and Gaussian Elimination, on a PC cluster. Our PC cluster has 16-node of PCs which are connected by a Gigabit Ethernet switching hub (Dell PowerConnect 5224) without jumbo frame support. Each PC has Xeon 2.8GHz, 1GB memory, Intel 82545GM Gigabit Ethernet controller, and Linux 2.6.8 running. We used MPICH 1.2.6 library for interprocess communication.

4.1 Application Programs

Jacobi Method is an iterative method for solving linear equations, such as $\mathbf{Ax}=\mathbf{b}$. When each processor updates subsection of $\mathbf{x}$ which is decomposed in block manner, the processor uses all other subsets of $\mathbf{x}$ which are updated by other processors at the previous step. For ordering the update and the use of data, a single barrier synchronization is inserted in the loop. In this evaluation, the number of iterations is 1,000 for any matrix size.

Gaussian Elimination also solves linear equations. In this evaluation, the matrix is decomposed in cyclic manner because the range of matrix updated narrows gradually with the progress of computation. At step $k+1$, a_{ij} is computed using the whole of row k, where $i > k$, $j > k$. In order to maintain this data dependency, a barrier synchronization is needed for each step.

In this evaluation, all elements of matrices are double-precision floating point values in both application programs.

4.2 Baseline Results

In this section, as baseline results, we show the sequential execution time and speedups of the programs before eliminating barrier synchronizations in running on our PC cluster. In order to measure the execution time of the main loop, Intel's RDTSC (Read Time Stamp Counter) instructions are inserted before and after the loop and then the difference between their counted value are computed.

Table 1. Sequential execution time in Jacobi Method and Gaussian Elimination

	Jacobi Method			Gaussian Elimination		
	Marix Size			Matrix Size		
	2,048	4,096	8,192	2,048	4,096	8,192
Exec. Time (sec)	42.15	167.79	670.02	13.96	108.56	852.77

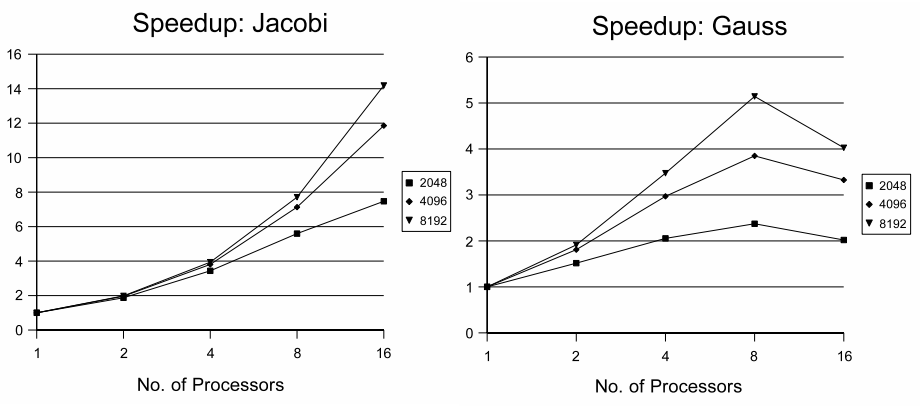


Fig. 7. Speedup of Jacobi Method and Gaussian Elimination before barrier elimination

Elapsed Time of Sequential Execution. Table 1 shows the sequential execution time of the two programs. We varied the size of matrix, denoted as N, from 2,048 to 8,192 in both programs. The execution time becomes approximately four times and 7.80 times in Jacobi Method and Gaussian Elimination respectively when the matrix size is doubled.

Speedups Before Barrier Elimination. Figure 7 shows the speedups of the programs before eliminating barrier synchronizations. We obtained 14.19-fold speedup for 16 processors in the case that N is 8,192 in Jacobi Method. On the other hand, in Gaussian Elimination, the speedup for 16 processors is worse than the one for 8 processors. The reasons for these results are as follows: 1) in Jacobi Method, achieving balanced load among the processors causes the high speedups, 2) in Gaussian Elimination, as mentioned above, the range of matrix updated narrows gradually with the progress of computation. This leads to imbalance of load among the processors, and 3) the number of barrier synchronizations issued at runtime in Gaussian Elimination (i.e., $N-1$) is more than the one in Jacobi Method (i.e., 1,000). This produces the overhead and results in less speedups.

4.3 The Effects of Barrier Elimination

For evaluation, we measured the number of barrier synchronizations eliminated and the reduction ratio of the execution time by applying our algorithm. We also measured idle time on a processor in detail.

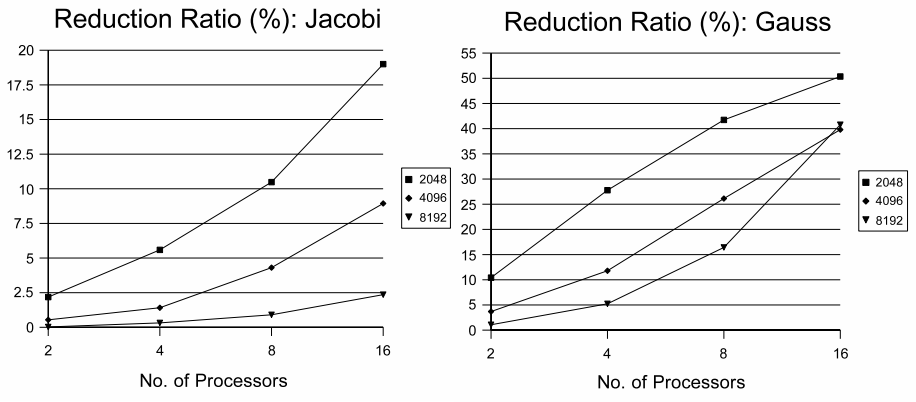


Fig. 8. Reduction ratio of execution time: Jacobi Method and Gaussian Elimination

The Number of Barrier Synchronizations Eliminated. In both programs, all barrier synchronizations which appear in the main loop are eliminated by applying our algorithm. Therefore, no processor is globally synchronized at runtime.

The Reduction Ratio of the Execution Time. We measured the reduction ratio of the execution time, which is computed as $100 \times (1 - T_e/T_b)$, where T_b and T_e are the execution time before and after eliminating barrier synchronizations respectively.

Figure 8 shows the plots of the reduction ratio. In both programs, 1) the reduction ratio is always positive and 2) the reduction ratio tends to increase as the number of processors increases.

In Jacobi Method, while the reduction ratio is 19.00% for 16 processors in the case that N is 2,048, the ratio is 2.36% when N is 8,192. Nevertheless, we obtained 14.53-fold speedup in this case.

On the other hand, in Gaussian Elimination, the reduction ratio is 50.36% and 40.79% for 16 processors in the case that N is 2,048 and 8,192 respectively. Speedup for 16 processors in the case that N is 8,192 is 6.18 and is improved as compared with the one for 8 processor, which is 6.15.

Frequency Distribution of Idle Time. If a processor must wait to the completion of the computations which are done by other processors when the processor finished to compute at a phase of a program, the processor becomes idle. We measured idle time, that is the elapsed time between the end of a phase and the beginning of the next phase after our algorithm applied and compared the idle time with the one in the case that barrier synchronizations are remained.

In Fig. 9, we show the frequency distribution of idle time for 16 processors in the case that N is 8,192 in both programs. In this evaluation, we used idle time which is obtained on Processor 0.

In Jacobi Method, the most of transitions between two consecutive phases completed within 5 msec even if barrier synchronizations are remained. However, while 1.2% of 1,000 transitions completed within 3.5 msec when barrier

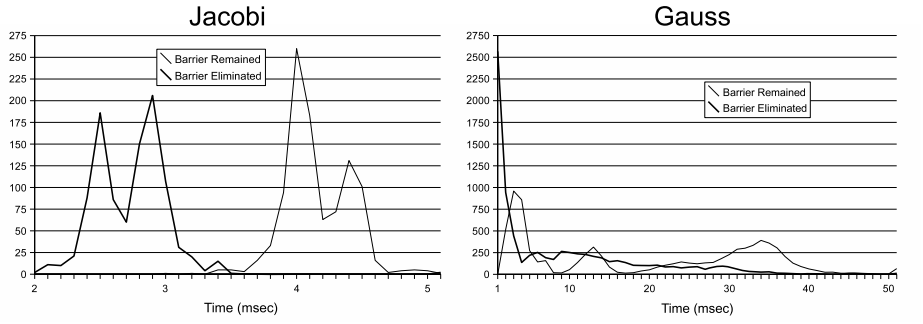


Fig. 9. Frequency distribution of idle time: Jacobi Method and Gaussian Elimination

synchronizations are remained, 99.80% of transitions completed after barrier synchronizations eliminated.

In Gaussian Elimination, no transition completed within 1 msec before barrier synchronization eliminated. On the other hand, after barrier synchronizations eliminated, 2,568 transitions (31.35%) completed within 1 msec. In the case that barrier synchronizations are remained, 5,503 transitions (67.18%) completed within 30 msec. If the barrier synchronizations are eliminated, 7,895 transition (96.39%) completed within 30 msec.

Discussion. In this evaluation, we found that 1) eliminating barrier synchronizations reduced the execution time in all cases in both programs, 2) the reduction ratio increases as the number of processors increases in all cases, and 3) the effectiveness of eliminating barrier synchronizations is influenced by the characteristics of application programs.

The reasons for the different effect between application programs are as follows:

- As mentioned above, the loads among the processors dynamically change in Gaussian Elimination. In such programs, a processor which waits for other processors at a phase can increase other processors' idle time at the next phase. If the barrier synchronizations are eliminated in the programs, the processor which completed the computation can send data immediately without waiting for other processors.
- Achieving load balance among processors in Jacobi Method tends to suppress idle time unless there is external factors such as network delay.

Nevertheless, in Jacobi Method, the most of idle time is less than 3.5 msec after barrier synchronizations eliminated because no communication with the barrier master occurs.

5 Related Works

Tseng proposed an algorithm for eliminating barrier synchronizations appear in SPMD programs [6] and evaluated it on Stanford DASH multiprocessor [8].

He also proposed another algorithm for replacing barrier synchronizations with lower overhead counter-based synchronizations. After replacing using this algorithm, all data-consumers wait until a data-producer completes to update a counter's value. This algorithm assumes the existence of shared memory which DASH provides.

Dwarkadas *et al.* proposed an algorithm for replacing barrier synchronizations with *push* function in which data-producer sends data to data-consumers [9]. They adopted Bounded Regular Section [10] as an array section descriptor while we adopted quad. For evaluation, they executed six application programs and the algorithm could be applied to two programs of them. They reported that the reductions of the execution time are limited. Additionally, they have not evaluated with varying the number of processors.

Some papers on Software DSM systems which are implemented using compiler analysis have been published in recent years [11,12]. However, any algorithms for eliminating barrier synchronizations are not implemented on their DSM systems and thus are not evaluated.

6 Conclusions

In this paper, we proposed the compiler algorithm for eliminating barrier synchronizations. It takes advantage of existence of data dependency as opposed to the conventional approach.

For evaluation, we executed two application programs on a PC cluster. For 16 processors, we obtained 19.00% and 50.36% of the reduction ratio of execution time for Jacobi Method and Gaussian Elimination respectively. With barrier elimination, 1) the execution time is always reduced, and 2) as the number of processors increases, the reduction ratio of the execution time also increases.

The results shown in this paper implies that we can obtain the more reduction ratio as the number of processor increases. We will evaluate the effectiveness of the proposed algorithm using a various kind of application programs on larger-scale clusters.

Acknowledgement

This research was supported by Japan Society for the Promotion of Science, a Grant-in-Aid for Scientific Research(B), No. 16300012.

References

1. OpenMP Architecture Review Board: OpenMP Application Program Interface. Version 2.5 edn. (2005)
2. Min, S.J., Basumallik, A., Eigenmann, R.: Supporting realistic OpenMP applications on a commodity cluster of workstations. In: Proceedings of the WOMPAT 2003. (2003) 170–179

3. Huang, L., Chapman, B., Liu, Z., Kendall, R.: Efficient translation of OpenMP to distributed memory. In: Proceedings of the ICCS 2004. (2004) 408–413
4. Eigenmann, R., Hoeflinger, J., Kuhn, R.H., Padua, D.A., Basumallik, A., Min, S.J., Zhu, J.: Is OpenMP for grids? In: Proceedings of the 16th International Parallel and Distributed Processing Symposium. (2002)
5. Yonezawa, N., Wada, K., Ogura, T.: *Quaver*: OpenMP compiler for clusters based on array section descriptor. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks. (2005) 234–239
6. Tseng, C.W.: Compiler optimizations for eliminating barrier synchronization. In: Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Santa Barbara, CA (1995)
7. Yonezawa, N., Wada, K.: *quad*: Array section descriptor for parallel computing. IPSJ Journal **46**(5) (2005) 1274–1286
8. Lenoski, D., et al.: The Stanford DASH multiprocessor. IEEE Computer **25**(3) (1992) 63–79
9. Dwarkadas, S., Cox, A.L., Zwaenepoel, W.: An integrated compile-time/run-time software distributed shared memory system. In: ASPLOS. (1996) 186–197
10. Havlak, P., Kennedy, K.: An implementation of interprocedural bounded regular section analysis. IEEE Transactions on Parallel and Distributed Systems **2**(3) (1991) 350–360
11. Min, S.J., Basumallik, A., Eigenmann, R.: Optimizing openMP programs on software distributed shared memory systems. International Journal of Parallel Programming **31**(3) (2003) 225–249
12. Manoj, N.P., Manjunath, K.V., Govindarajan, R.: CAS-DSM: A compiler assisted software distributed shared memory. International Journal of Parallel Programming **32**(2) (2004) 77–122

Parallelization Algorithms for Three-Body Interactions in Molecular Dynamics Simulation

Jianhui Li, Zhongwu Zhou, and Richard J. Sadus

Centre for Molecular Simulation
Swinburne University of Technology
PO Box 218, Hawthorn, Victoria 3122
Australia
{jli, zzhou, rsadus}@it.swin.edu.au

Abstract. Two force decomposition algorithms are proposed for parallelizing three-body interactions in Molecular Dynamics (MD) simulations. The first algorithm divides the entire 3D force matrix into equal sized force cubes that are assigned to parallel processors. In the second strategy, the force matrix is decomposed into slices of two-dimensional force matrixes, and those slices are distributed among processors cyclically. The proposed decomposition algorithms are implemented using MPI and tested in computational experiments. The performances of proposed decomposition methods are studied and compared with computational load theoretical analysis. Both theoretical prediction and computation experiments demonstrate that the load balance is a key factor that impacts the parallel performance of the examined system, and the cyclic force decomposition algorithm produced reasonably good overall performances.

1 Introduction

Molecular Dynamics (MD) simulation is a widely used computational tool to study fluid phenomena, many properties of molecular systems, and fabrication process of ultra-precision devices with dimensions down to sub-micro or nano level [1, 2]. Because of femtosecond step length and the large number of interactions that need to be evaluated, MD simulations are typically computationally intensive, and millions of times steps (wall time can be hours, days, or months long depending on the problem size and computer capacity) are necessary to simulate phenomena, which occur within several picoseconds. In this context, parallel computation techniques attracted much attention of scientists with its significant reduction of wall time by distributing massive computation among networked processors [3, 4]. The parallelism of MD simulation has been examined extensively [5-11]. A replicated-data method, also called atom decomposition, was proposed by W. Smith [6] and implemented in biological MD programs such as CHARMM and GROMOS. Force Decomposition methods based on strategies of decomposing force matrix were studied [7-9], and implemented (for execution on SIMD and MIMD machines). Spatial decomposition, also called geometric decomposition or domain decomposition as discussed in [10, 11], is widely used for very large MD system, where normally cutoff is significantly shorter than simulation box length. Compared to the spatial decomposition, force decomposition

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 374 – 382, 2006.
© Springer-Verlag Berlin Heidelberg 2006

displayed better performance in small to middle-sized problems where only short-range Van der Waals interactions are considered [9]. However, the previous studies were restricted to the parallelization of two-body interactions; none of them have yet investigated the parallelization of three-body interactions.

Three-body interactions play an indispensable role in reproducing many properties, such as phase equilibrium, second virial coefficient and etc.[12, 13]. The introduction of three-body interactions into MD simulations involves more complicated potential descriptions as well as a large amount of triplets to be evaluated, which imposes new challenge to the-state-of-the-art decomposition algorithms. Generally, three-body interactions imply significantly intensive computation even for small and middle-sized problems. In these types of problems, a potential cutoff is usually chosen to be or slightly shorter than half box length, therefore both atom decomposition and domain decomposition algorithms lose their strength. Former force decomposition algorithms proposed for pair-wise interactions encounter restriction for many-body forces as the force matrix is *m* rather than two-dimensional. Stimulated by the idea in the two-body force decomposition, a force decomposition for three-body interactions is firstly proposed, in which a three-dimensional force matrix is divided into sub force cubes. The second algorithm, cyclic force decomposition algorithm, is designed for efficiently parallelizing three-body interactions. To the best of our knowledge, these algorithms have not been attempted so far. The algorithms are implemented in our benchmark problem using MPI, and overall performances of the algorithms are evaluated in terms of load balance, speedup and parallel efficiency. Better parallelization performance with improved loading balance is achieved.

2 Computational Aspects of MD and Three-Body Interactions

In MD simulations, the system of *N* particulates evolves by integrating Newton's motion equations for all the particles in the system step by step, given by,

$$m_i \frac{d\mathbf{v}_i}{dt} = \mathbf{F}_i \qquad \mathbf{v}_i = \frac{d\mathbf{r}_i}{dt} \tag{1}$$

where m_i is the mass of atom i, $\mathbf{r}_i$ and $\mathbf{v}_i$ are its position and velocity vectors. The total force, $\mathbf{F}_i$, on the particle i due to interactions with other particles can be expressed with equation 2, ,in which $\mathbf{f}_{ij}$ is the force acting on the particle i due to the pair-wise interaction with particle j. The force term $\mathbf{f}_{ijk}$ is the contributory component on particle i by the three-body interactions with other two particles of I and j. Obviously, the force term $\mathbf{f}_{ijk}$ and $\mathbf{f}_{ikj}$ are exactly the same one and only one of them needs to be evaluated when the motion equations are integrated. High order many-body interactions may be further added for more precise evaluation. The most time-consuming part in MD simulation is force evaluation. The force terms in equation 2 are calculated as first derivatives of the potential energy existing among particles.

$$\mathbf{F}_i = \sum_j \mathbf{f}_{ij}(\mathbf{r}_i, \mathbf{r}_j) + \sum_j \sum_k \mathbf{f}_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k) + \cdots \tag{2}$$

Three-body forces are resulted from dispersion, induction and exchange effects that are identified as non-additive, and are important in reproducing many properties of condensed matters. In general, the force element can be expressed as:

$$\mathbf{f}_{ijk} = \nabla u_{ijk}(r_{ij}, r_{jk}, r_{ik}, \theta_i, \theta_j, \theta_k) \tag{3}$$

where r_{ij}, r_{ik}, and r_{jk} are side lengths of the triplet triangle consisting of particles i, j and k, and θ_i, θ_j, and θ_k are the three inside angles of the triangle, respectively. The actual force expressions vary from types of many-body interactions, and specific consideration of the model. Axilrod-Teller term [15] provides an expression of three-body dispersion potential as in equation 4, in which ν is non-additive coefficient.

$$u_{ijk} = \nu \cdot (1 + 3\cos\theta_i \cdot \cos\theta_j \cdot \cos\theta_k) / (r_{ij} \cdot r_{ik} \cdot r_{jk})^3 \tag{4}$$

For a system of N particles, the total number of unique triplets to be evaluated is $N(N-1)(N-2)/6$. Each single particle may be involved in $(N-1)(N-2)/2$ triplets. A triplet of particles of i, j, and k ($a \neq b \neq c$) contains 6 force elements. However, the force elements $\mathbf{f}_{ijk}$ and $\mathbf{f}_{ikj}$ are exactly the same, suggesting only three independent force elements are actually required to evaluate for each triplet. The computational cost of three-body interactions outweighs the computational cost of two-body interactions for the exponentially increase of interactions and its complex force expressions. In this paper only three-body force decomposition is addressed to emphasize our particular interest.

3 Decomposition Algorithms

3.1 Traditional Force Decomposition for Three-Body Interactions

Fig.1 schematically illustrates the traditional force decomposition algorithm for three-body interactions (FD-3). All three-body interactions form the outmost three-dimensional force matrix. When N is equally divided into m segments along three axes, total m^3 cubes are obtained and the equivalent number of processors is needed for the one cube-one processor decomposition. The inside solid cube holds three segments of particles from three dimensions, the a^{th} segment is from i dimension, the b^{th} segment from j dimension, and the c^{th} segment from k dimension. To distinguish one cube from another, the cubes are labeled with their contained segment indexes as subscripts. The sub force cube $\boldsymbol{F}_{abc}$, carries out the following calculations,

$$\vec{f}_{ijk} = \nabla u(\vec{r}_{ij}, \vec{r}_{jk}, \vec{r}_{ik}) \tag{5}$$

where $i \in a$, $j \in b$ and $k \in c$. Each cube has $(N/m)^3$ elements. To compute the sub force matrix $\boldsymbol{F}_{abc}$, the dedicated processor needs information from the (N_a, N_{a+1}), (N_b, N_{b+1}) and (N_c, N_{c+1}) segments. However, the computation load and memory requirement on each processor may vary depending on the contained segments of particles. In case of $a=b=c$, the required memory and computation load are proportional to N/m and $(N/m)(N/m-1)(N/m-2)/2$, respectively. When $a \neq b \neq c$, the corresponding values are $3*N/m$ and $(N/m)^3$.

For a triplet of i, j and k ($i \neq j \neq k$) only three independent force elements are actually required to evaluate in each step. The redundant calculation of the force elements $\boldsymbol{f}_{ijk}$ and $\boldsymbol{f}_{ikj}$ not only costs computation time but also causes errors in the integration of motion equations. A two-level 'check board' method is applied to keep unique calculation of each triplet. It works both at both cube and particle levels; at the cube level

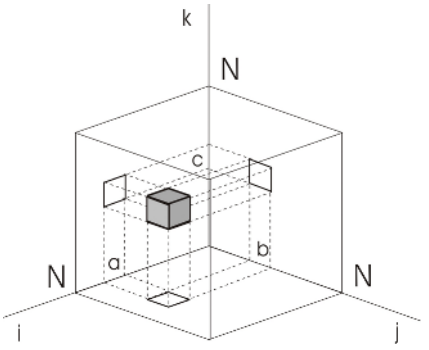


Fig. 1. FD-3 method

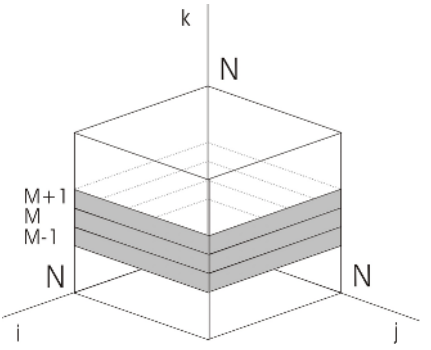


Fig. 2. CD-3 method

cubes that have identical particle segments are removed from job list. But those with unique combination of cube index *a, b* and *c* will be assigned for computation. We classify all the force cubes into three groups. The first group contains the same cube index, eg. $a=b=c$. The total number of these cubes is *m,* and *m* processors are needed. The second group includes those cubes with three different cube indices, eg. $a \neq b \neq c$,. The total number of cubes with this kind of combination is $m(m-1)(m-2)$. For three different values of *a, b* and *c*, there are 6 combinations, but only one is uniquely required for force evaluations, therefore these $m(m-1)(m-2)$ force cubes only need to be assigned to $m(m-1)(m-2)/6$ processors. The remaining $m^3-m-m(m-1)(m-2)$ cubes have two identical indices, such as $a=b\neq c$, and need $(m^3-m-m(m-1)(m-2))/3$ processors. A particle level check board technique works to remove any computational repetition in each sub force cube. For any force element $\boldsymbol{f}_{ijk}$ in a sub force matrix, it will be evaluated only in the case of $i\neq j\neq k$ and $j < k$, otherwise it will be assigned to zero.

1. Decompose three-body force matrix to cubes
2. Compute partial force matrix, $\boldsymbol{F}_{abc}$
3. Fold partial forces along k to get $\boldsymbol{F}_{ab}$
4. Fold partial forces along j to obtain $\boldsymbol{F}_{a}$
5. Update particle positions

Fig. 3. FD-3 Algorithm

The decomposition algorithm is outlined in Fig. 3. In step 1, the overall three-body force matrix is decomposed to m^3 sub force matrixes and unique force cubes are firstly identified. Proper particle segment data are copied to corresponding processors. In step 2, each processor carries out the assigned computation tasks (load) to obtain partial force matrix, $\boldsymbol{F}_{abc}$. All calculated partial force matrixes are then collected from each processor and sum over each row along *k* axis (eg. sum all force elements with the same *i* and same *j* index values) to fold the force matrix into a 2D matrix $\boldsymbol{F}_{ab}$ in step 3. A typical force element $\mathbf{f}_{ij}^{s}$ of the matrix $\boldsymbol{F}_{ab}$ can be expressed as:

$$\vec{f}_{ij}^{\,S} = \sum_{k=1}^{N} \vec{f}_{ijk} \qquad (i = 1 - N,\ j = 1 - N) \tag{6}$$

In step 4, the sum of the elements along the *j* axis in the matrix $\boldsymbol{F}_{ab}$ gives a force vector $\boldsymbol{F}_{a}$ with elements expressed by equation 7. Each element in the force vector $\boldsymbol{F}_{a}$

represents the total three-body force acting from all possible triplets of involving particles.

$$\vec{f}_i^{\,S} = \sum_{j=1}^{N} \vec{f}_{ij}^{\,S} \qquad (i = 1 - N) \tag{7}$$

The individual computation load can be predicted by the assigned triplets, so do the achieved balance status. With our theoretical prediction, maximal computation load appears on the cubes that have unequal segments of particles, while the least computation intensive ones are those on diagonals. And higher imbalance might happen with force decomposition method.

3.2 Cyclic Force Decomposition Algorithm

Fig.2 depicts the proposed cyclic force decomposition algorithm. This strategy divides the matrix into force slices along one dimension, in our case, k direction. Each slice consists of a 2D force matrix. Slice m evaluates all triplets formed by m with any other two particles, such as l and n in the system. These force slices are then distributed among processors according to rank order in a cyclic task assignment. There are totally N slices, and at any given moment each processor may dedicate itself to more than one slice either consecutively or inconsecutively. During one job distribution cycle each processor is assigned one slice, whenever there are still job slices left, another assignment cycle initializes. All job slices are given to working machines in a cyclic order of rank index. In this work for the convenience of notation, this strategy is also called CD-3.

The total number of triplets in one slice is $(N\text{-}1)(N\text{-}2)$. However, in the slice m, the triplets of (m, l, n) and (m, n, l) are exactly one. Therefore only half of the unique triplets are needed to be evaluated in one slice. Another redundancy exists as one given triplet can appear in three slices, for example, a triplet formed by l, m and n particles can appear in slices l, m and n. To avoid calculation repetition, a slice-level checking board method is applied to assign the triplet of l, m and n to the slice with the minimum index of l, m and n. This method ensures one triplet only to be evaluated in one slice, eventually total triplets to be evaluated in the slice m becomes $(N\text{-}m\text{-}1)(N\text{-}m\text{-}2)/2$. To rule out the excessive calculation within a triplet, a triplet-level check board technique is applied. In brief, for any force element $\mathbf{f}_{ijk}$ in a triplet, it will be evaluated only in the case of $i \neq j \neq k$ and $j<k$, otherwise it will be assigned to zero.

The proposed decomposition algorithm is outlined in Fig. 4. In the first step, the rank of dedicated processor for i^{th} slice is determined by the following rule, rank = module(i, N_p) and then followed by job slice assigning. Here N_p is the number of processors available. The partial sum forces calculated in step 2 need to be broadcasted to all other processors in step 3, then complete force vector that contains the total

1. Cyclically assign force slices among processors.
2. Compute force elements and partial sum of forces.
3. Broadcast partial force among all processors.
4. Force summation to obtain the complete force.
5. Update particle positions.

Fig. 4. CD-3 algorithm

three-body forces for all the particles are calculated in step 4 by summing all the partial sum forces. The complete forces are used in step 5 for updating particle positions. Updating is carried out on all the processors to avoid additional communication time and load imbalance.

The communications take place in step 3, and communication overhead is estimated as up to *O(N)* level. Memory storage requirement scales to *O(N)* as well, as the processor need to keep the entire information of the system, but this is not a big issue with the shared-memory systems which is often installed with gigabytes of RAM.

4 Results and Discussions

4.1 Benchmark

There are 500 water molecules in the benchmark NVT ensemble, where both two-body and three-body interactions are evaluated. The SHAKE [16] constraint algorithm is employed to keep water molecules rigid in our simulations. MCY potential [12], together with Axillrod-Teller three-body term is chosen. Computational experiments are conducted on Aqua Linux cluster in Swinburne University of Technology. Ninety-six Dell Optiplex GX280 computers with Intel Pentium 4 3.2GHz Hyperthreaded processors are connected, each two of them connect to the total 48 Gigabit ports, CISCO Catalyst 6509 Network switch is used with the fast forwarding blades. The networked computers have 1 GB of DDR2 RAM as its self-contained infrastructure. The operation system is Linux 2.6.8-24.14, on which LAM7.0.6 is running to provide parallel computation environment. An in house developed molecular dynamics C++ code was used for parallelization. The parallel decomposition algorithm is implemented in accordance with LAM C++ binding protocol.

MD simulations are carried out and timed on 4, 10, 20 and 35 CPU schemes, respectively. Each scheme was evaluated with the average of 4 runs on the same machine setup to remove unexpected impact from system operation. In each run, the system evolves 200 time steps, and computation loads are measured as computation time on each processor averaged from the 5^{th} step to the 195^{th} step for all machines. The beginning and ending of simulation are removed from time sampling for their involvement of I/O operation and memory administration, which delay the computation. Timing was conducted by logging computation progress on each processor in specially designed code segment, which can hook the system time during computation.

4.2 Load Balance

The load balance status is studied in term of step time variation ratio (*STVR*), which is defined as in equation 9.

$$STVR_i = \left| \frac{T_i - (\sum_{j=1}^{N_p} T_j)/N_p}{(\sum_{j=1}^{N_p} T_j)/N_p} \right| \tag{9}$$

The average step computation time, *Ti*, is measured on the processor *i*, which is directly related to the total triplets evaluated on it. Since the intensive three-body interactions are parallelizable, the execution time of serial code can be neglected. In our benchmark tests, the communication overheads happening at the end of each time step are ruled out, but only "pure three-body computation time" is measured. Therefore, the obtained STVR is the index of imbalance contributed by decomposition algorithms.

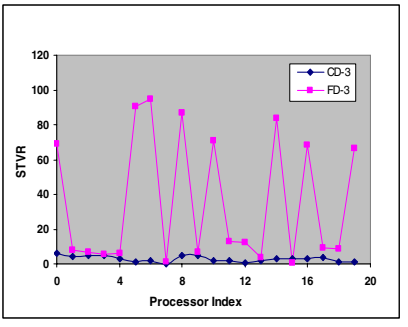


Fig. 5. Load imbalance on 20 CPUs

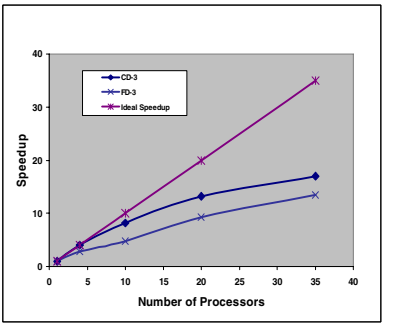


Fig. 6. Speedups on processors

Fig. 5 presents an example of loading imbalance obtained from simulation experiments on 20 processors. The maximal *STVR* data obtained from benchmark measurements on 4, 10, 20 and 35 CPUs are listed in Table 1. The proposed decomposing method CD-3 shows a significantly better loading balance than the FD-3, but both of them with an increasing tendency of *STVR*, as the number of processors increases.

Table 1. Maximum *STVR* for the proposed decomposition algorithms

CPUs	4	10	20	35
*STVR**	0.6/100	1.79/178	3.58/157	6.26/144
*STVR***	0.76/50	4.5/126	6.1/90	13.9/79

* Experimental results, CD-3/FD-3; ** Theoretical prediction, CD-3/FD-3

The computation load on each processor is roughly proportional to the assigned total number of triplets. The total triplets number N_T on a processor P_i can be evaluated by summing the assigned triplets over all assigning cycles as expressed by equation 10, where C_i is the number in the i^{th} cycle. Therefore the achieved load balance can be theoretically predicted by equation 10. The predicted maximal computation load should appear on processor 0, while the least computation loads on the last processor. Equation 10 [14] also suggests the load difference between these two processors could become larger as more processors (larger N_p) are used, and hence higher imbalance might happen. As shown in Table 1, both experimental and theoretical calculation results display a similar increasing trend of imbalance with increase of processors. However, the theoretical prediction gave much higher imbalance value

than the experiment, especially when many processors are applied. This might result from the supplementary code for matrix decomposing, job slice assigning, and their influences are not taken into account in the prediction.

$$N_T(P_i) = \sum_{C_i=0}^{\mathrm{mod}\left(\frac{N}{N_p}\right)-1} \frac{(N - P_i - C_i \cdot N_p - 1) \cdot (N - P_i - C_i \cdot N_p - 2)}{2} \tag{10}$$

We conducted the theoretical analysis of force decomposition on three-body force matrix and found that this would result in severe load imbalance problem. In the computational experiment, extreme high imbalance is observed with FD-3 algorithm, and the resultant STVR index is significantly higher than that with CD-3. Our proposed cyclic force decomposition algorithm provides considerable improvement in the load balance, particularly for the three-body interactions, although further improvement still needs to be sought.

4.3 Parallel Performance

The overall parallel performance of the algorithm was evaluated in terms of speedup. The achieved speedup results for DF-3 and CD-3 are shown in Fig. 6. Speedup of 3.9, 8.2, 13.2 and 17.1 were achieved when 4, 10, 20 and 35 processors were used for CD-3 algorithm respectively, but 2.75, 4.72, 9.28, and 13.5 for FD-3 algorithm. Many factors impact the overall speedups, such as, serial code percentage, communication overhead, memory access, and loading imbalance. The serial code part limits the maximum achievable speedup [9]. However, for our system examined in this study, the serial code part is actually negligible, and the communication overheads in different schemes are also small. As discussed in 4.2, the loading imbalance increases with the increase of processors employed. It is evident that the loading imbalance is a major obstacle for achieving a high parallel efficiency although other factors need to be quantified in future work. The imbalance problem of force decomposition had been discussed for two-body pair interactions by Plimton [3, 9], but its quantitative impact on the parallel performance of three-body interactions has not been analyzed until this work.

5 Conclusions

In this study, FD-3 and CD-3 decomposition algorithm have been proposed and implemented to parallelize three-body interactions in MD simulations. The proposed decomposition algorithms are based on the decomposition of a three-dimensional force matrix. Relatively good balance status and parallelization performances are achieved with CD-3 algorithm through benchmark experiments on our specific MD systems. The theoretical analysis of the force computation load on each processor provided a theoretical prediction of a similar increasing trend of imbalance with increasing processor number to the benchmark measurement results. Both theoretical analysis and computation experiments demonstrate that the load imbalance is a key factor that impacts the parallel efficiency of three-body interactions.

References

1. Bosko, J.T., Todd, B. D., Sadus, R. J.: Journal of Chemical Physics, 121 (2004) 1091-1096
2. Rentsch, R., Brinksmeier, E., Li. J.: Nonlinear Dynamics of Production Systems, Viley-VCH (2003) 245-263.
3. Plimpton, S.: Journal of Computational Physics 117 (1995) 1-19
4. Roy, S., Jin, R. Y., Chaudhary, V., Hase, W. L.: Computer Physics Communications 128(2000) 210-218
5. Schreiber, H., Steinhauser, O., Schuster, P.: Parallel Computing 18 (1992) 557-573
6. Smith, W.: Computer Physics Communications 62 (1991) 229-248
7. Boyer, L L., Pawley, G. S.: Journal of Computational Physics 78 (1988) 405-423
8. Brunet, J. P., Edelman, A., Mesirov, J. P.: SIAM Journal of Scientific and Statistical Computing 14 (1993) 5 1143-1158
9. Plimpton, S.: Journal of Computational Chemistry 17 (1996) 3 326-337
10. Fincham, D.: Molecular Simulation 1 (1987) 1-45
11. Gupta, S.: Computer Physics Communications 70 (1992) 243-270
12. Matsuoka, O., Clement, E., Yoshimine, M.: Journal of Chemical Physics 64 (1976) 4 13511361
13. Marcelli G., Sadus, R.J.: Journal of Chemical Physics 111 (1999) 1533-1540
14. Li, J., Zhou, Z., Sadus, R. J.: “Modified Force Decomposition Strategies for Three-Body Interactions in Molecular Dynamics Simulations”, to be published on Computer Physics Communications (2006)
15. Axilrod, B.M., Teller, E.: Journal of Chemical Physics 11(1943) 6 299-300.
16. Sadus, R. J.: “Molecular Simulation of Fluids, Theory, Algorithms and Object-Orientation”, Elsevier, Amsterdam, (2002)

An Efficient Distributed Algorithm for Mining Association Rules

Zahra Farzanyar[1], Mohammadreza Kangavari[2], and Sattar Hashemi[2]

[1] SECOMP Lab., Department of Computer & IT, Iran University of Science & Technology (IUST), Tehran, Iran
nsfarzan@yahoo.com
[2] Department of Computer & IT, Iran University of Science & Technology (IUST), Tehran, Iran
kangavari@iust.ac.ir, s_hashemi@iust.ac.ir

Abstract. Association Rule Mining (ARM) is an active data mining research area. However, most ARM algorithms cater to a centralized environment where no external communication is required. Distributed Association Rule Mining (DARM) algorithms aim to generate rules from different datasets spread over various geographical sites; hence, they require external communications throughout the entire processor. A direct application of sequential algorithms to distributed databases is not effective, because it requires a large amount of communication overhead. DARM algorithms must reduce communication costs. In this paper, a new solution is proposed to reduce the size of message exchanges. Our solution also reduces the size of average transactions and datasets that leads to reduction of scan time, which is very effective in increasing the performance of the proposed algorithm. Our performance study shows that this solution has a better performance over the direct application of a typical sequential algorithm.

Keywords: Distributed Data Mining, Association Rules, Distributed Databases.

1 Introduction

ARM is an important research area in the field of data mining. The goal of ARM is to discover associations between attribute values. Association rules can be rated by a number of quality measures, among which support and confidence stand out as the two essential ones [22, 23]. On the basis of Apriori algorithm, there are two phases for mining association rules [1]. The key work for finding the association rules is to find all the frequent itemsets.

Because databases are increasing in terms of both dimension (number of attributes) and size (number of records), one of the main issues in a frequent itemset mining algorithm is the ability to analyze very large databases [4]. Sequential algorithms do not have this ability, especially in terms of run-time performance, for such very large databases. Therefore, we must rely on high performance parallel and distributed computing.

Parallelism is expected to relieve current ARM methods from the sequential bottleneck, providing the ability to scale to massive datasets, and improving the

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 383 – 393, 2006.
© Springer-Verlag Berlin Heidelberg 2006

response time. Most existing parallel and distributed ARM algorithms are based on a kernel that employs the well-known Apriori algorithm [2, 3 and 6]. Directly adapting an Apriori algorithm won't significantly improve performance over frequent itemsets generation or overall DARM performance. To perform better than Apriori algorithms, we must focus on their problems. The main challenges include synchronization, communication minimization, work-load balancing, finding good data layout and data decomposition, and disk I/O minimization, which is especially important for DARM. In distributed mining, synchronization is implicit in message passing, so the goal becomes communication optimization. Data decomposition is very important for distributed memory. Therefore, the main challenge for obtaining good performance on distributed mining is to find a good data decomposition among the nodes for good load balancing, and to minimize communication. In this paper, we have developed a distributed algorithm for geographically distributed datasets that reduces communication costs by focusing on similar behaving attributes.

This paper is organized as follows: Section 2 summarizes the related work in distributed ARM. Section 3 investigates the definitions and methodology of DARM also it introduces a formula to testing strong dependence between attributes and describes a new algorithm of Distributed mining association rules. Section 4 shows the experimental results. The last section concludes the paper.

2 Related Work

Several parallel and distributed versions of sequential algorithms for ARM have been proposed in the last years [2, 5, 7, 10, 12, 15, 16, 17, 18 and 19]. The major difference between parallel algorithms is in the architecture of the parallel machine. This may be shared memory as in the case of [8, 13], distributed shared memory as in [11], or shared nothing as in [2, 10, 12].

Parallelism in both shared-memory and distributed memory ARM algorithms can be categorized as data-parallelism or task parallelism [1, 14]. Data parallelism logically partitions the database among processors. Each processor works on its local partition of the database, but performs the same computation of counting support for the global candidate itemsets. Data-parallelism exchanges only the counts among processors, which minimizes the communication cost. Data-parallelism is however only suitable for use on homogenous databases. Count Distribution (CD) is a simple data-parallelism algorithm [2]. CD algorithm uses the sequential Apriori algorithm in a parallel environment and assumes datasets are horizontally partitioned among different sites. Each processor generates the local candidate itemsets independently based on the local partition. Then the global counts are computed by sharing (or broadcasting) the local counts, and the global frequent candidate itemsets are generated. The focus of the CD algorithm is on minimizing communication. This algorithm has a simple communication scheme for count exchange. However, it also has the similar problems of higher number of candidate sets and larger amount of communication overhead. It does not use the memory of the system effectively.

In task-parallelism, each processor performs different computations independently, yet all have access to the entire dataset. Data Distribution (DD) is a task-parallelism-based algorithm that partitions the candidate itemsets among the processors [2]. It

thus suffers from high communication overhead and performs poorly when compared with CD algorithm. Hybrid parallelism combining both task and data parallelism is also possible. Candidate Distribution algorithm [2] combining both CD and DD is presented. The CD algorithm performed the best among the three algorithms. It exhibited linear scale-up and excellent speed-up and size-up behavior. Several algorithms have been proposed to reduce the communication load in the CD algorithm. Distributed algorithms (DMA, FDM) are presented in [23, 10] which generate fewer candidates than CD, and use effective pruning techniques to minimize the messages for the support exchange step. For the communication, instead of broadcasting the local counts of all candidates as in CD, they send the local counts to polling site. FDM's message optimization techniques require some functions to determine the polling site, which could cause extra computational cost when each site has numerous local frequent itemsets. Furthermore, each polling site must send a request to remote sites other than the originator site to find an itemset's global support counts, increasing message size when numerous remote sites exist. Ashrafi, et al. [20] propose the Optimized Distributed Association Mining (ODAM) algorithm based on CD which both reduces the size of the average transaction and reduces the number of message exchanges in order to achieve better performance. As for the message exchange, instead of using broadcast as with CD or polling sites like FDM, ODAM just sends all local information to one site, called the receiver. The receiver then calculates the global frequent itemsets and sends them back. Based on our survey, In CD-based distributed algorithms, each local site generate support counts and broadcasts them to all other sites which causes a huge amount of communication. Algorithms such as FDM, DMA and ODAM reduce communication costs by assigning one site to act as a server to calculate global frequencies. Up to the point where the central server becomes the bottleneck. This kind of mechanism requires, however, that some sites rely on others (usually only one) to obtain the final ruleset. This dependency is avoided in our distributed algorithm.

3 DARM and Proposed Algorithm

Communication is one of the most important DARM objectives. DARM algorithms will perform better if we can reduce communication costs (for example, message exchange size). To reduce communication costs, we take message optimization techniques. We can divide the message optimization techniques into two methods direct and indirect support counts exchange [20]. The first method exchanges each candidate itemset's support count to generate globally frequent itemsets of that pass (CD and FDM are examples of this approach). The second method to reduce communication costs, doesn't consider an itemset's exact global support [12]. To maintain an association rule's correctness and compactness, our algorithm sticks with the first approach.

In this paper we present a new algorithm for share-nothing machines where each of n processors has a private memory and a private disk. The processors are connected by a communication network and can communicate only by passing messages. Data is evenly distributed on the disks attached to the processors.

3.1 Problem Definition

Let $I = \{i_1, i_2, ..., i_m\}$ be the set of items. Let DB be a database with D transactions. Assume that there are n sites $s_1, s_2, ..., s_n$ in a distributed system and the database DB is partitioned over the n sites into $\{DB_1, DB_2, ...DB_n\}$, respectively. Let the size of the partitions DB_i be D_i, for $i = 1,2,...,n$. Let $X.\sup$ and $X.\sup_i$ be the support counts of an itemset X in DB and DB_i, respectively. $X.\sup$ is called the global support count, and $X.\sup_i$ the local support count of X at site s_i. For a given minimum support threshold s, X is globally frequent if $X.\sup \geq s \times D$. The essential task of a distributed ARM algorithm is to find the globally frequent itemsets L.

Our proposed idea in this algorithm in order to reduce communication overhead is fusing the similar behaving attributes. For this propose, we need a measure of dependency between two items. A widely used one has been introduced in [21], it is chi-square and is based on support difference.

3.2 Chi-Square Test for Independence and Correlation

Chi-square test statistics (χ^2) is a widely used method for testing independence and/or correlation [21]. In our proposed technique, it is used for testing strong dependence between attributes. Below, we give an introduction to chi-square test. Essentially, χ^2 test is based on the comparison of observed frequencies with the corresponding expected frequencies.

Example: In a loan application domain, we have 500 people who applied for loan in a bank. Out of the 500 people, 300 had a job and 200 did not have a job. 280 people were granted loan and 220 were not. We also know that 200 people who had a job were granted loan, which can also be expressed as an association rule:

$$\text{Job} = \text{yes} \rightarrow \text{Loan} = \text{approved} \quad [\text{Sup} = 200/500, \text{conf} = 200/300]$$

This information gives us a 2×2 *contingency table* containing four cells (Table 1). Note that the table has only 1 degree of freedom, which is sufficient for binary attributes [21].

Table 1. Contingency table for Job and Loan

	loan=approved	loan=not-approved	Row Total:
job=yes	200	100	300
job=No	80	120	200
column Total:	280	220	500

Our question is “Is loan approval dependent on whether one has a job or not?” In dealing with this problem we set up the hypothesis that the two attributes are

independent. We then compute the expected frequency for each cell as follows: Of the 500 people included in the sample, 300 (60% of the total) had a job, while 200 (40% of the total) had no job. If the two attributes are truly independent, we would expect the 280 approved cases to be divided between job = yes and job = no in the same ratio; similarly, we would expect the 220 not-approved cases to be divided in the same fashion.

χ^2 is used to test the significance of the deviation from the expected values. Let f_0 be an observed frequency, f be an expected frequency, r be each cell in the table, and R be the set of all cells. The χ^2 value is defined as:

$$\chi^2 = \sum_{r \in R} \frac{(f_0 - f)^2}{f} \tag{1}$$

A χ^2 value of 0 implies the attributes are statistically independent. If it is higher than certain threshold value (e.g. 3.84 at the 95% significance level [21]), we reject the independence assumption. For our example, we obtain χ^2 = 34.63. This value is much larger than 3.84. The independence assumption is rejected. Thus, we say that the loan approval is correlated to (or dependent on) whether one has a job.

3.3 Proposed Algorithm in Detail

DARM algorithms must reduce communication costs so that generating global frequent itemsets costs less than combining the participating sites' datasets into a centralized site. However, most DARM algorithms don't have an efficient message optimization technique, so they exchange numerous messages during the mining process. Some of them reduce communication costs by assigning one site to act as a server to calculate global frequencies where the central server becomes the bottleneck.

Our algorithm offers better performance by minimizing candidate itemset generation costs. In this method the similar behaving global frequent attributes are found and fused. Such attributes are presented together in more of the transactions. Finding two similar behaving attributes in one set and separate studying of each is something time-consuming and leads to communication overhead. In our proposed algorithm, we have used Chi-square statistic test to study the similar behaving of attributes.

To deal with this problem, we propose a new distributed algorithm based on CD algorithm [2] that fragments the dataset into different horizontal partitions. The algorithm first computes support counts of 1-itemsets from each site in the same manner as it does for the sequential Apriori. It then broadcasts those itemsets to other sites and discovers the global frequent 1-itemsets.

To efficiently generate candidate support counts of later passes, the algorithm fuses all of the similar behaving global frequent 1-itemsets after the first pass and places those new transactions into the main memory. For this purpose, each site constructs contingency tables for each of two global frequent 1-itemsets. Then, each site obtains observed frequencies for all of the contingency tables at the same time by scanning once the local partition. They send contingency tables to a single site. We refer to this site as the coordinator. Then, the coordinator constructs contingency tables related to each of two attributes entirely. For each of these tables, the coordinator is responsible

for obtaining that expected frequencies in case of accepting the independence assumption, and finally it calculates the χ^2 value. The χ^2 value obtained from the chi-square test for each of the contingency tables, is compared to the chi-square table by the coordinator and the similar behaving global frequent 1-itemsets are determined. After obtaining all the similar behaving global frequent 1-itemsets, considering the common points between them, the coordinator deduces the final similar behaving global frequent 1-itemsets and broadcasts them to all of the sites. Then, Each processor fuses the final similar behaving global frequent 1-itemsets independently on the primary local partition and inserts the new transaction into memory. For example, A and B are similar behaving attributes, therefore in all transactions, A and B is fused and appears in the form of A-B. (In the phase of AR making, we will show the behavior of similar behaving attributes as Meta rule to maintain exactness of the obtained rules. Such rules are the main rules and they show the behavior of similar behaving attributes.).

Nevertheless, when we fuse similar behaving attributes of each transaction, the size of each transaction is reduced and the chance of finding similar transactions increases. During the writing, a tag is placed in front of every transaction to specify how many times that transaction exists in a partition. While inserting the new transaction, the algorithm checks whether that transaction is already in the memory. If it is, it increases that transaction's counter by one. Otherwise, it inserts the transaction with a count equal to one into the main memory. Finally, it writes all main-memory entries for this partition into a temp file. This process continues for all other partitions. This technique reduces the average transaction length and also reduces the dataset size significantly, so we can accumulate more transactions in the main memory. These two reductions in transactions size and dataset size lead to reduction of scan time, which is very effective in increasing the performance of the proposed algorithm.

After fusing similar behaving attributes from each partition, the algorithm iterates through the new dataset (that is, the temp file) and generates the globally frequent itemsets of various lengths by broadcasting the support counts of candidate itemsets after every pass. It should be noted that the design of this step follows that of the Apriori data-mining algorithm.

In our algorithm the total number of candidate itemsets, to be generated, is much less than the CD algorithm and it causes the considerable reduction of the size of message exchange and also the time spent on computing the frequency of candidate itemsets. Thus we reduced communication costs. In algorithms such as FDM, DMA and ODAM, the central server becomes the bottleneck. This kind of mechanism requires, however, that some sites rely on others (usually only one) to obtain the final ruleset. This dependency is avoided in our distributed algorithm. The following shows the details of the proposed algorithm:

Procedure Distributed Mining of Association rules algorithm

Input: (1) DB_i: the database partition at each site; (2) s: the minimum support threshold; both submitted at each site S_i, (i= 1,…,n);

Output: L: the set of all global frequent itemsets in DB, returned at every site;
Method: iterates the following program fragment distributively at each site S_i starting from k =1, where k is the iteration loop counter; the algorithm terminates when either L_k returned is empty or the set of candidate sets C_k is empty.

Each site $\mathbf{S}_i$ generates $X.sup_i (X \in C_1)$ by scanning D_i;
/* C_1: candidate 1-itemsets */
Broadcast $(X.sup_i (X \in C_1))$;
Receive $(X.sup_i (X \in C_1)$ from all other sites s_i;
For all $X \in C_1$ do {

$$X.sup = \sum_{i=1}^{n} X.sup_i ;$$

If $X.Sup \geq S \times D$
Insert (X,X.sup) into L_1;} /* L_1: frequent 1-itemsets */

Each site $\mathbf{S}_i$, For all 2-subsets Y of L_1 generates contingency-table$_i$(Y) by scanning D_i;
Send-to-coordinator (contingency-table$_i$(Y));
Sim=Receive-from-coordinator (sim_m);
*/*Receiving similar behaving global frequent 1-itemsets from coordinator*/*
For all transactions $t \in D_i$ {
For all $sim_m \in sim$
t=Fuse (sim_m);}

For j=2 to k at each site $\mathbf{S}_i$ do {
While ($L_{k-1} \neq 0$) {
C_k = Apriori-Gen (L_{k-1})
For all transaction $t \in D_i$
For all k-subsets X of t
If ($X \in C_k$) $X.sup_i$++;}
Broadcast $(X.sup_i (X \in C_k))$;
Receive $(X.sup_i (X \in C_k))$ from all other sites s_i;
For all $X \in C_k$ do {

$$X.sup = \sum_{i=1}^{n} X.sup_i ;$$

If $X.Sup \geq S \times T$
Insert (X,X.sup) into L_k;} Return L_k; K++ ;}

4 Experimental Results

To illustrate the effect of fusing similar behaving attributes, we compare the performance of our new algorithm with the CD algorithm [2] and consider the superiority of it with the other algorithms such as FDM, DMA and ODAM. The CD algorithm is an adaptation of the Apriori algorithm in the distributed case. However, it has the similar problems of higher number of candidate sets and larger amount of communication overhead. We have extensively studied our algorithm's performance to confirm its effectiveness. We implemented this algorithm using C++. We established a socket-based, client-server distributed environment to evaluate our algorithm's message reduction techniques. Each site has a receiving and a sending unit and assigns a specific port to send and receive candidate support counts. Because the candidate itemsets that each site generates will be based on the global frequent itemset for the previous pass, the candidate itemsets are identical among various sites.

The datasets used for this performance study are the connect-4 dense dataset and the Cover Type dataset. Connect-4 contains many frequent itemsets also for high support thresholds, and the Cover Type is relatively sparse and uses low support thresholds to generate frequent itemsets. Both data sets are taken from the UCI datasets [24]. Table 2 shows the characteristics of each dataset.

Table 2. Data set characteristics

Name	Number of items	Number of records	Average transaction size
Connect-4	130	67,557	43
Cover type	120	581,012	55

To show how our algorithm can efficiently generate support counts, we conduct an experiment in a single site. Because the total number of sites is equal to one, CD will perform the same as the sequential Apriori algorithm. Figure 1 shows our algorithm and CD's total execution time for generating the frequent itemsets using the UCI datasets. As Figure 1 shows, our algorithm outperforms CD in all cases. However, our algorithm fuses a significant number of the similar behaving global frequent 1-itemsets from every transaction, so it finds a significant number of identical transactions. It requires a minimal number of comparison and update operations to generate support because it doesn't enumerate candidate itemsets multiple times for any identical transaction. In contrast, CD takes longer, thus requiring numerous comparison and update operations to generate support counts. We can observe that the executing time of our new algorithm is much less than the CD algorithm.

To compare the total communication cost (that is, the total size of messages exchanged of this algorithm and CD) between different sites to generate the globally frequent itemsets, we partition the original data set into four partitions. Each one contains only 25 percent of the original data set's transactions. So, the number of identical transactions among different partitions is very low. In the CD algorithm, each local site generates support counts and broadcasts them to all other sites to let each site calculate globally frequent itemsets for that pass. So, the total number of messages broadcast from each site equals $((n - 1) * |C|)$. We can calculate the total message size for that pass using

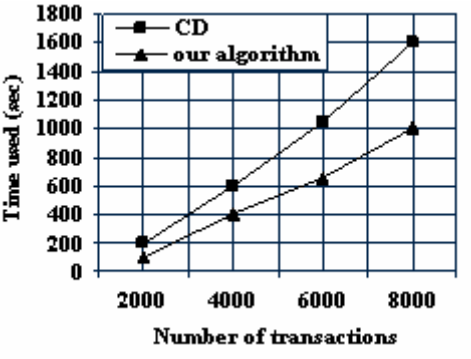


Fig. 1. Our algorithm versus the CD algorithm in a single node

$$T_{messages} = \sum_{i=1}^{n} (n-1) * |C| \tag{2}$$

Where n is the total number of sites and $|C|$ is number of candidate itemsets for that pass [2]. In our algorithm, the total message size for that pass is

$$T_{messages} = \sum_{i=1}^{n} (n-1) * |C'| \tag{3}$$

Where $|C'|$ and n are the number of candidate itemsets and the number of sites, respectively, ($|C'| << |C|$). Advantage of our algorithm over CD is that it reduces the communication cost by the reducing the total number of candidate itemsets. It generates fewer candidate itemsets compared to CD.

In the testing step of the similar behaving attributes, the total number of messages that each site sends to the coordinator and the coordinator sends to all other sites equal 2n.

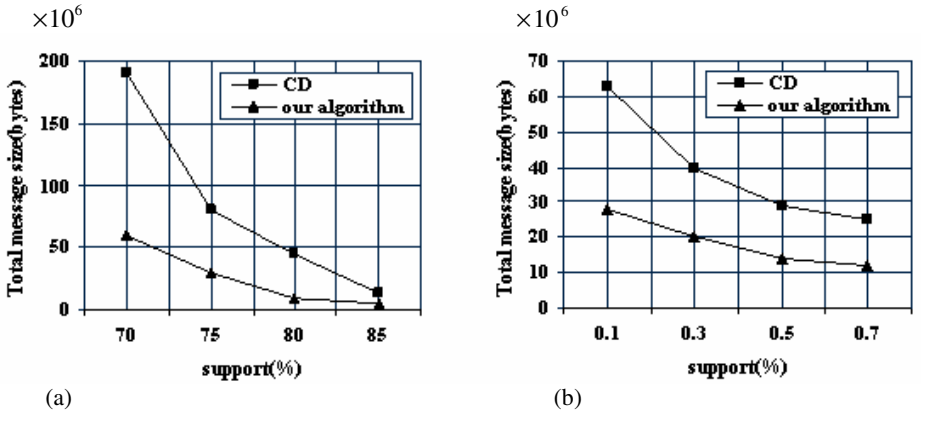


Fig. 2. Total message size that our algorithm and CD transmit to generate the globally frequent itemsets for (a) Connect-4 data and (b) Cover Type data

Therefore, this number of messages, in comparison with reducing the total number of candidate itemsets is trivial number. Figure 2 depicts the total size of messages (that is, number of bytes) that our algorithm and CD transmit to generate the globally frequent itemsets with different support values.

Our algorithm doesn't assign one site to act as a server to calculate global frequencies. Therefore, the scalability of our algorithm is increased. This is the superiority of our algorithm as compared with the other algorithms such as FDM, DMA and ODMA.

5 Conclusion

Association Rule Mining (ARM) is an important research area in the field of data mining. Parallelism is an ideal way to scale ARM to large databases. Distributed Association Rule Mining (DARM) algorithms must reduce communication costs so that generating global association rules costs less than combining the participating sites' datasets into a centralized site. We have developed an efficient algorithm for mining association rules in distributed databases. The developed method has the following impacts by recognition and fusion similar behaving global frequent 1-itemsets:

1- Reduces the size of message Exchanges
2- Reduces the size of average transactions and original datasets that leads to reduction of scan time, which is very effective in increasing the performance of the proposed algorithm.

This efficiency is approved by the experimental results. In algorithms such as FDM, DMA and ODAM, the central server becomes the bottleneck. This kind of mechanism requires, however, that some sites rely on others (usually only one) to obtain the final ruleset. This dependency is avoided in our distributed algorithm. Therefore, the scalability of our algorithm is increased.

References

1. R. Agrawal and R. Srikant , "Fast Algorithms for Mining Association Rules in Large Database,"Proc. 20th Int'l Conf. Very Large Databases (VLDB 94), Morgan Kaufmann, 1994. 40
2. R. Agrawal and J.C. Shafer , "Parallel Mining of Association Rules,"IEEE Tran. Knowledge and Data Eng., vol. 8, no. 6, 1996, pp. 962-969; http://csdl.computer.org
3. R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast Discovery of Association Rules in Large Databases. In Advances in Knowledge Discovery and Data Mining, pages 307{328. AAAI Press, 1996.
4. A. Savasere, E. Omiecinski, and S.B. Navathe , "An Efficient Algorithm for Mining Association Rules in Large Databases, "Proc. 21st Int'l Conf. Very Large Databases (VLDB 94), Morgan Kaufmann, 1995, pp. 432-444.
5. J. Han, J. Pei, and Y. Yin , "Mining Frequent Patterns without Candidate Generation,"Proc. ACM SIGMOD Int'l. Conf. Management of data , ACM Press, 2000,pp. 1-12.

6. M.J. Zaki and Y. Pin , "Introduction: Recent Developments in Parallel and Distributed Data Mining,"J. Distributed and Parallel Databases , vol. 11, no. 2, 2002,pp. 123-127.
7. M.J. Zaki , "Scalable Algorithms for Association Mining,"IEEE Trans. Knowledge and Data Eng. , vol. 12 no. 2, 2000,pp.372-390; http://csdl.computer.org
8. M.J. Zaki , et al., Parallel Data Mining for Association Rules on Shared-Memory Multiprocessors , tech. report TR 618, Computer Science Dept., Univ. of Rochester, 1996.
9. D. W. Cheung, Vincent T. Ng, Ada W. Fu and Y. Fu, Efficient Mining of Association Rules in Distributed Databases, IEEE transactions on Knowledge and Data Engineerin, 1996.
10. D.W. Cheung, et al., "A Fast Distributed Algorithm for Mining Association Rules," Proc. Parallel and Distributed Information Systems, IEEE CS Press, 1996, pp. 31-42; http://csdl.computer.org/comp/proceedings/pdis/1996/7475/00/74750031abs.htm.
11. Z. Jarai, A. Virmani, and L. Iftode. Towards a cost-effective parallel data mining approach. Workshop on High Performance Data Mining (held in conjunction with IPPS'98), 1998.
12. A. Schuster and R. Wolff. Communication-efficient distributed mining of association rules. In Proc. Of the 2001 ACM SIGMOD Int'l. Conference on Management of Data, pages 473 ñ 484, Santa Barbara, California, May 2001.
13. R O. R. Zaiane, M. El-Hajj, and P. Lu. Fast parallel association rules mining without candidacy generation. In IEEE 2001 International Conference on Data Mining (ICDM'2001).
14. V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining Very Large Databases. IEEE , 1999.
15. E-H. S. Han, G.Karypis, and V.Kumar. Scalable parallel data mining for association rules. In IEEE Trans action on Knowledge and Data Engineering, 2000.
16. Shintani and Kitsuregawa. Hash based parallel algorithms for mining association rules. In PDIS: International Conference on Parallel and Distributed Information Systems. IEEE Computer Society Technical Committee on Data Engineering, and ACM SIGMOD,1996.
17. S.Orlando, P.Palmerini, R.Perego, and F.Silvestri. A scalable multi-strategy algorithm for counting frequent sets. 5th Int. Workshop on High Perf. Data Mining HPDM, 2002.
18. Mohammed J. Zaki. Parallel and distributed association mining: In IEEE Concurrency, 1999.
19. A.Schuster, R.Wol_, and Dan Trock. A high-performance distributed algorithm for mining association rules. In Proc. IEEE International Conference on Data Mining ICDM'03, 2003.
20. M.Z Ashrafi, *Monash University* ODAM: An Optimized Distributed Association Rule Mining Algorithm, IEEE DISTRIBUTED SYSTEMS ONLINE 1541-4922 © 2004 Published by the IEEE Computer Society Vol. 5, No. 3; March 2004 F.Mills, *Statistical Methods*, Pitman, 1955.
21. J.S Park: Ming-Syan Chen and Philip S. Yu IBM Thomas J.Watson, Efficient Parallel Data Mining for Association Rules Research Center Yorktown Heights,New York 10598
22. J.S. Park, M. Chen, and P.S. Yu, "An Effective Hash Based Algorithm for Mining Association Rules,"Proc. 1995 ACM SIGMOD Int'l Conf. Management of Data, ACM Press, 1995.
23. J.W. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, San Francisco, (2001).
24. C.L. Blake and C.J. Merz , UCI Repository of Machine Learning Databases, Dept. of Information and Computer Science, University of California, Irvine, 1998;

Adaptive Algorithms Using Bounded Memory Are Inherently Non-uniform

Burkhard Englert*

California State University Long Beach, Dept. of Comp. Engr. & Comp. Science
Long Beach, CA 90840
benglert@csulb.edu

Abstract. Distributed protocols that run in dynamic environments such as the Internet are often not able to use an upper bound on the number of potentially participating processes. In these settings *adaptive* and *uniform* algorithms are desirable where the step complexity of all operations is a function of the number of concurrently participating processes (adaptive) and the algorithm does not need to know an upper bound on the number of participating processes (uniform). Adaptive algorithms, however, are generally not adaptive with respect to their memory consumption - if no upper bound on the number of participating processes is known in advance - they require unbounded MWMR registers and an unbounded number of such registers (even if only finitely many distinct processes appear), making them impractical for real systems. In this paper we ask whether this must be the case: Can adaptive algorithms where no upper bound on the number of participating processes is known in advance be uniformly implemented with finite memory (if only finitely many distinct processes keep reappearing)? We will show that in the dynamic setting it is impossible to implement long-lived adaptive splitters, collect and renaming with infinitely many **bounded** MWMR registers, making such adaptive algorithms impractical in dynamic settings. On the positive side we provide algorithms that implement a long-lived uniform adaptive splitter if unbounded registers are available and that implement a non-uniform adaptive splitter with finitely many bounded registers if an upper bound on the number of participating processes is known in advance.

1 Introduction

Many well known and important distributed algorithms such as atomic snapshot or renaming, require processes to gather information about each other. For example, in the renaming problem, before choosing a unique new name, processes need to know which names other processes have already chosen. To communicate this information an array of Single-Writer Multi-Reader (SWMR) registers can be used. Each process has a unique array entry assigned to it and only a fixed process is allowed to write to each array location while all processes can read

* Research supported by the Dept. of Transportation under METRANS USC-111699.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 394–405, 2006.
© Springer-Verlag Berlin Heidelberg 2006

them. A process can *update* information about itself by writing into its entry and it can then *collect* information about the other processes by reading all entries in an arbitrary order.

Such a collect algorithm with step complexity $O(N)$, however, where N is the total number of processes in the system, is possibly inefficient or impractical if only few of the N processes are actually participating or if no upper bound on the number of participating processes is known in advance. This and other similar issues motivated researchers to look for *adaptive* and *uniform* algorithms whose step complexity only depends on the number of concurrently participating processes (adaptive) and that don't need to know an upper bound on the number of participating processes in advance (*uniform*).

Adaptive algorithms have a worst case step complexity that is bounded by a function of the number of concurrently participating, or actually active processes [3]. Motivated by Lamport's MX algorithm [18], many such adaptive algorithms have since been designed [1,3,4,5,10,11,14,19]. The strongest forms of adaptiveness in the read/write shared memory model have been defined and achieved in recently presented long-lived adaptive collect [6] and renaming [1,12] algorithms. In these algorithms, called *adaptive to point contention* the number of steps taken by a process executing an operation is a function of the maximum number of processes that were active simultaneously at some point in time during this operations execution interval. Algorithms *adaptive to interval contention* have a slightly weaker level of adaptiveness. Here the number of steps taken during an operation is a function of the total number of different processes active during this operations execution interval. Finally, an algorithm is adaptive to *total contention* if the number of steps taken by a process is a function of the total number of processes active since the beginning of the execution.

In all these algorithms [1,3,4,5,6,7,11,12,13,15,16,19,20], however, the memory consumption is a function of the upper bound N on the number of processes that **might** participate in the algorithm. Moreover these algorithms usually assume **unbounded** MWMR registers. They are not concerned with a distributed system such as the Internet, where we have a potentially huge number of processes that might participate in some protocol but it is known that with very high probability only a small number of processes will be active at any given time or participate. It is unrealistic and wasteful for such a system to provide a huge number of shared memory registers for the operation of such a protocol. Also in a real system unbounded MWMR registers are not available. Algorithms that operate in a dynamic setting are not able to use a priori knowledge about a finite upper bound on the number of processes in the system and are called *uniform* algorithms. Aguilera, Englert and Gafni [8] showed that there are single shot tasks such as generalized weak test and set [9] that cannot be solved uniformly with a finite number of MWMR registers. In other words a protocol solving this task with finitely many MWMR registers must know the number of participating processes in advance. Since generalized weak test and set is a one-shot algorithm, this implies that the long lived nature of test and set and the requirement that the step complexity adapt to interval contention are not the only requirements that preclude a solution in finite space.

So far all known adaptive algorithms (e.g. [1,3,4,5,6,7,11,12,13,15,16,19,20]) require knowledge of an upper bound on the number of participating processes. This upper bound must be hard coded into the algorithm, making them non-uniform. They are not adaptive with respect to their memory consumption. To summarize, this lack of adaptiveness and non-uniformness has two aspects.

1. Unbounded MWMR registers are needed, greatly limiting their usability in real systems.
2. The algorithms must a priori know N, an upper bound on the number of participating processes.

These algorithms are non-uniform since for each possible number of participating processes a "different" algorithm is required.

It would be desirable to have uniform algorithms that can "on the fly" adjust to the number of participating processes in such a way that no matter how many are actually participating, as long as the number is finite, the algorithm - even in an unbounded execution will use only finitely many bounded MWMR registers. To our knowledge, no such algorithm exists.

So we ask if this apparent lack of uniformity and reliance on unbounded memory is an inherent property of adaptive algorithms. What is the relationship between uniform, adaptive algorithms and bounded memory? To get an answer to this question we will investigate whether it is possible to have truly adaptive algorithms using bounded memory in a setting where no upper bound N on the number of participating processes is known. In an execution where only finitely many distinct processes appear can we implement adaptive algorithms using finitely many **bounded** MWMR registers?

In other words are there uniform adaptive algorithms that can use bounded memory in infinite executions? This is an important question since its answer has strong implications on the practicality of adaptive algorithms in dynamic settings.

Table 1. Implementability of long-lived, adaptive splitter, collect, renaming

	Implementation possible
∞-many bounded reg., uniform impl.	No
Fin. many bounded reg, N known, non-uniform impl.	Yes
∞-many unbounded reg., uniform impl.	Yes

We will first show that even if we provide infinitely many bounded MWMR registers it is impossible to implement weak test & set in an execution where N is unknown. By reduction this implies that the use of at least finitely many unbounded registers is a necessary ingredient for uniform and adaptive implementations of long lived and adaptive splitter, collect and renaming. In other words, adaptive algorithms that want to be uniform must either use unbounded registers as a black box or have some a priori knowledge about the upper bound of possibly participating processes. This makes them impractical in dynamic settings. As it turns out they require a closed and controlled environment to be

truly effective. On the positive side we present a long-lived, uniform and adaptive to interval contention implementation of a splitter that in an unbounded execution (where only finitely many distinct processes appear) uses infinitely many unbounded MWMR registers. We finally show how this algorithm can be modified to use bounded memory if an upper bound on the number of participating processes is known in advance (making the resulting algorithm non-uniform). The results are summarized in Table 1.

1.1 Adaptive Splitter

The splitter that we will implement is an essential building block of many of these adaptive renaming, snapshot and collect algorithms(e.g. [1,3,4,5,10,11,14,19]). Splitters were first introduced by Moir and Anderson [20]. A splitter is a variant of mutual exclusion. If one process accesses the splitter alone it captures the splitter and the resource (name) associated with it. When several processes access the splitter concurrently they may all fail leaving the splitter un-captured. The splitter wait-freely partitions the processes that fail into two groups, *right* and *down*. At the same time the splitter ensures that not all contending processes go down and not all go to the right. If splitters are put into a grid then processes that begin in the top left are split into smaller and smaller groups as they traverse the splitter until individual processes access a splitter on their own, guaranteeing them success. Our splitter will have fewer properties than the Moir Anderson [20] splitter but it has a long-lived and adaptive implementation.

1.2 Related Work

Englert and Goldstein [17] recently considered protocols that they called *memory-adaptive*. In such protocols processes are **at all times** only allowed to write to MWMR registers whose **index is a function of the contention during the previous shared memory write operation**. They showed that in a system with *infinitely* many MWMR registers and *infinitely* many SWMR registers, for any constant d, there exists a number N_d such that if N_d processes are allowed to participate in a *memory-adaptive* (to POINT contention) execution of the protocol, then at least one does not make a single uncovered write to a shared register in d writes. In other words they showed that under these conditions processes cannot memory-adaptively store a value in shared memory. This implies that any time-adaptive collect or renaming algorithm in this setting that uses only finitely many of the infinitely many MWMR registers (if such an algorithm exists) cannot be built from memory-adaptive building blocks alone.

Note that our model used in this paper is more general than the model in [17]. We allow processes to write to registers with any index. We only require that in an unbounded execution - as long as only finitely many processes appear - only finitely many such registers are actually written to. Hence the results presented in [17] do not answer the question whether there exists a long-lived, step complexity-adaptive (not necessarily memory-adaptive) and uniform renaming or collect protocol that uses only finitely many bounded MWMR registers.

1.3 Contributions and Paper Organization

In Section 2 we introduce our model. Our contributions are as follows.

I. In Section 3 we show that using only bounded shared MWMR registers there is no adaptive, uniform implementation of long-lived splitter, collect and renaming in an unbounded execution with finitely many processes appearing.
II. In Section 4 we provide a uniform algorithm adaptive to interval contention that implements a splitter and uses infinitely many unbounded MWMR registers in an unbounded execution.
III. In Section 5 we illustrate that using a previously presented algorithm [7] and given a bound on N the largest id of participating processes we can implement a non-uniform, long-lived, adaptive splitter with **finitely many bounded** MWMR registers (in an execution where only finitely many processes participate).

We conclude with some final remarks in Section 6.

2 Model and Preliminaries

Our algorithms assume an asynchronous read/write shared memory model. This model consists of a set of N asynchronous processes $p_0, p_1, ..., p_{N-1}$ and a set of registers shared by the processes. The processes communicate only through the registers which provide two atomic operations, read and write. We assume single-writer, multi-reader (SWMR) and multi-writer, multi-reader registers (MWMR). A bounded register of size M is a register that can hold at most M distinct values. An *unbounded* register is a register for which no such bound exists. A protocol is called *long-lived* if processes are able to execute it infinitely often. Algorithms that are not able to use a priori knowledge about a finite upper bound on the number of processes in the system are called *uniform* algorithms.

Let α be an execution of a long-lived algorithm A and α' a prefix of α. Process p_i is *participating* at the end of α', if α' includes an invocation of some operation of A by process p_i without the matching response. The *active processes* at the end of α', are denoted $Cont(\alpha')$. It is the set of processes participating at the end of α'. Given a subsequence β of α, let $\alpha'\beta$ be the shortest prefix of α that contains β. We define the *interval contention* denoted $\text{IntCont}(\beta)$ and *point contention* denoted $\text{PntCont}(\beta)$ as follows:

$$\text{IntCont}(\beta) = |\bigcup_{\alpha'\beta' \text{ prefix of } \alpha'\beta} Cont(\alpha'\beta')|$$

$$\underset{\alpha'\beta' \text{ prefix of } \alpha'\beta}{\text{PntCont}(\beta) = \max}|\text{Cont}(\alpha'\beta')|$$

Intuitively the interval contention of a subsequence β is the number of different processes that were participating during β, while the point contention is the maximum number of processes active at any point in time during β. Clearly $\text{PntCont}(\beta) \leq \text{IntCont}(\beta)$.

We define the *total contention* of any execution α as the number of processes that took steps in α.

Let *op* be an operation. We define the *execution interval* of *op*, denoted $\beta(op)$ as the subsequence of α starting at the invocation of *op* and ending at the completion of *op*. The interval contention of an operation *op* is defined as $\text{IntCont}(\beta(op))$. In the rest of the paper, k denotes $\text{IntCont}(\beta(op))$ for some operation *op*. The step complexity of an algorithm is *adaptive to interval contention* if there is a bounded function S, such that the number of steps performed by any process p_i in any execution interval of an operation op_i of A is at most $S(IntCont(\beta(op_i)))$. Clearly the contention of an execution interval is bounded by N, the total number of processes participating in the algorithm. We will assume that this bound is a priori unknown, that is that our algorithms have no knowledge of this bound in advance. Hence, in an adaptive algorithm with bounded concurrency any operation op_i (by p_i) terminates with a bounded number of steps, regardless of the actions of any other processes (than p_i). So any adaptive algorithm is by definition *wait-free*.

Informally a splitter is a weak mutual exclusion primitive. Processes access the splitter through an *invoke* operation. This operation has three possible responses: stop, right or down. A process that receives a *stop* response has captured the splitter. Such a process releases the splitter by invoking a release operation that has only one possible response: done.

In this paper we will only consider well-formed operations where no process has more than one pending operation at any given point in time and where a process only invokes a release operation if and only if its last event was a stop response.

For every process p we define the *p-active* intervals with respect to the splitter execution to be: 1. From an acquire invocation of the splitter by p until the corresponding response if the splitter was not captured by p. 2. From an acquire invocation of the splitter by p until the done response to the corresponding release invocation, given that the splitter was captured by p.

A process that has no acquire operations cannot be *p-active* and is hence called *p-idle*. We will implement an adaptive splitter with the following properties:

1. At any point in time the adaptive splitter is captured by at most one process. (Mutual exclusion)
2. If the prefix of a busy period is only an invocation of acquire and its response, then the response must be a stop. (Processes that access a "new" splitter by themselves must capture it.)
3. Not all responses to the acquire invocations during a busy period are right.
4. Not all responses to the acquire invocations during a busy period are down.
5. No busy period of the adaptive splitter that contains infinitely many events where the events are only acquire invocations and down responses (i.e. where no process captures it or goes right).

As always with such splitters, if two or more processes access it concurrently it is possible that none of them captures it. But not all acquire attempts return down and not all return right.

The key difference between our splitter and the Moir Anderson [20] splitter are the following two properties: 1. At the point in time where a process is guaranteed to go right there is at least another process active in the splitter that might either stop or go down. 2. At the point in time where a process is guaranteed to go down there is at least another process active in the splitter that is guaranteed not to go down or is undecided (might go down, right or stop).

3 Impossibility of Uniform, Adaptive Splitter, Collect and Renaming with Infinitely Many Bounded Registers

We now show that it is impossible to uniformly implement adaptive long-lived splitter, collect or renaming using infinitely many bounded MWMR registers in an unbounded execution where at most finitely many distinct processes may appear. To do so we use a so called *weak test and set* object [2].

3.1 Weak Test and Set

We model the behavior of a weak test and set object with the following program (Figure 1). Each process is in one of four possible states: thinking, WT & Set, eating and RESET.

```
TS: object of type WT&S
Process p:
repeat forever
    thinking_section_p
    tbit:= WT&SET_p(TS)
    if tbit = 0 {
            eating_section_p
            RESET_p(TS) }
end repeat forever
```

Fig. 1. Weak Test & Set algorithm

A WT&S object satisfies the following two properties:

- **Exclusion:** At most one process is eating at any system state of the execution.
- If a process becomes hungry, that is leaves the thinking state, while all other processes are thinking and it only takes steps then it must eventually start eating.

Clearly if we can implement a long-lived adaptive splitter, we can implement WT&SET. The reduction is straightforward, we simply use the adaptive splitter to implement the WT&SET object. A process that captures the splitter enters the eating section. Processes that fail to capture the splitter do not eat. This algorithm has the required properties, it implements WT&SET. Moreover as was shown in [2] both a long-lived adaptive collect or a long-lived adaptive renaming algorithm can be used to implement the WT&SET object. Hence it suffices to show that we cannot implement an adaptive WT&SET object using only register reads and writes in an unbounded execution where finitely many distinct processes may appear and infinitely many bounded registers are available.

Theorem 1. *There does not exist a uniform, long-lived, adaptive implementation of a Weak-Test & Set object using only read and write operations, infinitely many bounded MWMR registers and infinitely many bounded SWMR registers in an execution where only finitely many distinct processes appear.*

We first restate a combinatorial lemma proved in [17].

Lemma 1. *Let $\mathcal{N}$ be the set of all integers. Assume that you are given an infinite collection of sets $S_i \subseteq \mathcal{N}$ such that $|S_i| \leq k$ for some nonnegative constant k. Then $\exists X \subseteq \mathcal{N}$ such that $|X| = \infty$ and such that $\forall x, y \in X, y \neq x \Rightarrow x \notin S_y$.*

The proof of this lemma can be found in [17]. Processes are potentially able to write information about themselves into their own SWMR registers. This information could then subsequently be read by other processes "helping" them in their execution of the WT&SET algorithm. The sets S_y here represent the ids of processes corresponding to the SWMR registers that process y is going to read. Hence informally speaking the lemma says that there is an infinite set of processes X such that no two processes in X read each others SWMR registers when executing the WT&SET algorithm while believing they are running alone.

Using Lemma 1, the idea of the proof of Theorem 1 is to show that we can get at least two carefully selected processes p and q to execute the algorithm together in such a way that these two processes will not be able to distinguish each others writes. In other words p will interpret a write of q it reads as one of its own writes and vice versa. As a result they will both capture the WT&SET bit (eat) at the same time, a contradiction.

We first show that we can find an infinite set of processes that will all write to the same MWMR register. Note that we will not require that all these infinitely many processes execute the algorithm. They will simply provide us with the collection of processes from which we will be able to carefully select the finitely many processes with the desired properties.

Lemma 2. *There exists an infinite set $\mathcal{S}$ of processes that write to the same MWMR register and that do not read any register to which only finitely many of the processes in $\mathcal{S}$ wrote to.*

Proof Sketch: By Lemma 1, $\exists X_0 \subseteq \mathcal{N}$ such that $|X_0| = \infty$ and such that $\forall x, y \in X_0, y \neq x \Rightarrow x \notin S_y^{(0)}$. This implies that there exists an infinite set of processes X_0 such that no process reads the single-writer register of any other process in the set. Assume now that the processes in X_0 write to infinitely many distinct MWMR registers in their first write. Then we can intuitively think of these infinitely many MWMR as SWMR registers (where the last process writing to a register is the single writer) and apply Lemma 1 one more time - we get an infinite set of processes $X_1 \subseteq X_0$ with the same properties as X_0 (no process reads information of any other participating process). If processes continue to write to infinitely many distinct MWMR registers we can inductively continue this construction. Since our algorithm must be adaptive, a process that believes to execute the algorithm alone must terminate within a constant number k of

steps. Hence we obtain an infinite set of processes X_k that all finish the execution of the algorithm believing they ran solo and hence capturing the WT&SET bit, a contradiction. Hence **infinitely many processes** will eventually have to write to **finitely many MWMR registers**. So there must exist an infinite set $\mathcal{S}$ of processes that writes to the same MWMR register and where no process in $\mathcal{S}$ reads information about any other process in $\mathcal{S}$ in any other register. □

Using induction we now finish the proof of the Theorem:

We will construct an infinite set of processes that when executing the algorithm at the same time, write the same values to the same MWMR registers in the same order and do not read each others SWMR registers.

By Lemma 2 there must be an infinite set of processes X_0, where no two processes read each others SWMR registers and that write to the same bounded MWMR register r_1. By the boundedness of r_1 and the pigeonhole principle it follows that there is an infinite set of processes $X_1 \subseteq X_0$ such that none of the processes in X_1 is able to distinguish its write to r_1 from a write from one of the other processes in X_1. If the processes in X_1 finish their execution after writing to r_1 we are done. Otherwise by Lemma 2 there exists an infinite subset X_2 of X_1 of processes such that they all write to the same register r_2, do not read each others SWMR registers (by Lemma 2) and hence are not aware of each other. Since each of these processes believes it executes the algorithm on its own and since the algorithm is adaptive, there exists a k such that after k writes each of these processes must terminate its execution. Given k we can inductively continue this construction until we obtain X_k, an infinite set of processes that when executing the algorithm at the same time, write the same values to the same MWMR registers in the same order and do not read each others SWMR registers or any other register that could provide them information about each other. We now simply select two processes p and q from X_k and let them execute the algorithm at the same time. Since both p and q are members of X_k, that is, they write the same values to the same registers in the same order they will both capture the WT&SET bit at the same time, a contradiction. □

4 Adaptive, Long-Lived and Uniform Splitter Using Unboundedly Many Unbounded MWMR Registers

We now present an algorithm that implements an adaptive, long-lived splitter in a system where no upper bound on the number of participating processes is known. The algorithm assumes that at most finitely many processes will participate in the algorithm and uses infinitely many unbounded MWMR registers. It is based on [7].

We present the implementation of the long-lived adaptive splitter in Figure 2. Note that in the given construction a process that comes alone captures the adaptive splitter in $O(1)$ steps. It also releases the splitter in one step by setting its status to *idle*. However when k processes access the splitter concurrently some process may perform k steps. For example, let process p and q execute

```
acquire() for process p returns stop, down or right.
Type: pid = process id, 0, ....
Shared:
    X[0...], Infinitely many unbounded atomic registers of type pid each initialized to 0.
    Y[0...], Infinitely many unbounded atomic registers of type (pid each initialized to 0.
    Z[0...], Infinitely many unbounded atomic registers of type pid each initialized to 0.
    status[0...], Infinitely many atomic registers of type {start, active, idle} initialized to idle.
    I[0...], Infinitely many unbounded atomic registers consisting of i,
    where i is an integer 0 ≤ i initialized to −1.
    Master, an integer in the range 0, ... initialized to 0.
Code:
1   status[p] :=start; //indicate status
2   m := Master; //Master is supposed to be last process to update index
3   c := I[m]; //find last copy of splitter used
4   current := c + 1; // try next copy of splitter
5   if ((status[(X[c])] = active or (status[Y[c]] = active or (status[(Z[c])] = active then
            status[p] := idle; return right
6   status[p] := active; // set state in single-shot splitter.
7   X[current] := p; //emulate single shot splitter.
8   if Y[current] ≠ 0 then
            status[p] := idle, return right
9   I[p] := c; //Once p writes in Y[current], another process q may read it later. If I[p] is not updated this process will loop.
10  Y[current] := p, nextDB;
11  if (X[current] ≠ p) then
            update(current), status[p] := idle, return down
12  Z[current] := p; // make sure to be seen after capturing splitter.
13  if (X[current] ≠ p) then
            update(current), status[p] := idle, return down
14  update(current)
15  return stop
Procedure release() for process p
16  status[p] := idle
17  return;
Procedure update(c) for process p.
18  I[p] := c;
19  Master := p;
do forever
20  if (Master ≠ p ) then return // another process will update pointer, p can return.
21  c := c + 1;
22  if (Y[c] = 0 then return; //fresh copy found, p can return.
23  q := Y[c];
24  if (I[q] ≠ (i − 1) then c := I[q], I[p] := c; // only update I[p] if it is smaller than I[q].
    od;
```

Fig. 2. Implementation of adaptive and long-lived splitter

in lock-step until both are about to write to X. Assume that p now writes to X and that then q immediately overwrites p in X. Let them then execute in lock-step again until they read X. Assume that p now reads X and finds the condition to fail and stops right before performing *update(c)*. Process q can now continue and either capture the splitter or leave. While p is still active but not performing any steps k different processes can capture and release single shot copies of the splitter. Hence to finish its execution p will now have to iterate k times through the loop in *update()*, Figure 2. Since k is at most $N-1$, however the step complexity of p will still be $O(N)$.

We say that a process captures the adaptive splitter object (Figure 2) if it reaches line 14 of the algorithm. We first show that if a process reaches line 14, no other process will reach this line until the first process finished its execution. I.e. we show that our splitter has a standard mutual exclusion property. The proof is based on the correctness proof by Afek et al. [7] of their adaptive splitter implementation. Out of lack of space we simply quote the main result. We begin by showing that no more than one process at a time can capture the splitter.

Lemma 3. *No two processes p and q reach line 14 of the code concurrently.*

Proof. We leave the proof to the full version of the paper.

Lemma 4. *Any process executing the algorithm in exclusion must reach line 15.*

Proof. Same as in [7].

Theorem 2. *For a system with finitely many active processes there exists a long-lived adaptive uniform splitter implementation using infinitely many unbounded MWMR registers.*

5 Adaptive and Long-Lived Splitter Implementation Using Bounded Registers, Given N

Given N, an upper bound on the id's of processes that will appear during an infinite execution of the long-lived adaptive splitter, the previous algorithm can be modified so that it uses only finitely many bounded registers. Moreover, bounded MWMR and SWMR registers can be used instead of unbounded registers by estimating the size of registers required based on N. The resulting algorithm is the same as presented in [7]. Its main feature is that by assigning unique registers to processes in advance it allows us to reuse registers that processes wrote to before. Every time before accessing a new copy of the adaptive splitter, a process checks whether the process assigned to this copy is currently active. If so, the process does not write to this copy but simply "walks away", i.e. returns "right". This prevents a process p from writing to copies of the splitter where active processes are currently about to write to one of the registers, hence possibly erasing all traces of this process p. We are only able to do this since we know an upper bound on process id's in advance, hence are able to assign three unique MWMR registers to all possibly participating processes. Since moreover processes write only their id's and smaller values to all registers we are also able to bound the size of the registers in advance.

6 Discussion

We showed that any uniform, long-lived adaptive implementation of a splitter, collect or renaming cannot use only bounded MWMR registers if no a priori upper bound on the number of participating processes is known, even if only finitely many processes participate in the algorithm. This means that any such adaptive algorithm must use unbounded MWMR registers. Hence even if finitely many such registers would be sufficient unbounded memory is needed, making these step complexity adaptive algorithms impractical in dynamic settings. This shows that adaptive algorithms that run on real systems are inherently non-uniform. They are inherently designed for closed and controlled settings: Designers must find a way to realistically limit the size of the system so they can hard code an upper bound on the size of the system into the algorithm.

References

1. Y. Afek, H. Attiya, A. Fouren, G. Stupp and D. Touitou. Long-Lived Renaming made adaptive. *Proc. of 18th ACM Symp. on Principles of Distributed Computing (PODC)*: 91-103, 1999.
2. Y. Afek, P. Boxer and D. Touitou. Bounds on the shared memory requirements for long-lived and adaptive objects. *Proc. of the 19th ACM Symp. on Principles of Distributed Computing (PODC)*:81-89, 2000.
3. Y. Afek, D. Dauber and D. Touitou. Wait-free made fast. *Proc. of the 27th Ann. ACM Symp. on Theory of Computing*: 538-547, 1995.
4. Y. Afek and M. Merritt. Fast, wait-free $(2k-1)$-renaming. In *Proc. of the 18th Ann. ACM Symp. on Principles of Distributed Computing*: 105-112, 1999.
5. Y. Afek, M. Merritt, G. Taubenfeld and D. Touitou. Disentangling multi-object operations. In *Proc. of 16th Annual ACM Symp. on Principles of Distributed Computing*: 111-120, 1997.
6. Y. Afek, G. Stupp and D. Touitou. Long-lived adaptive collect with applications. *Proc. of the 40th Ann. Symp. on Foundations of Computer Science*: 262-272, 1999.
7. Y. Afek, G. Stupp and D. Touitou. Long Lived Adaptive Splitter and Applications. *Distributed Computing*, 15(2): 67-86, 2002.
8. M. Aguilera, B. Englert and E. Gafni. Uniform Solvability with a finite number of MWMR registers. In *Proc. 17th International Conference DISC 2003*: 16-30, 2003.
9. J. Anderson and J-H. Yang. Time/contention trade-offs for multiprocessor synchronization. *Information and Computation*, 124(1):68-84, 1996.
10. H. Attiya and E. Dagan. Universal operations: Unary versus binary. In *Proc. 15th ACM Symp. on Principles of Distributed Computing*: 223-232, 1996.
11. H. Attiya and A. Fouren. Adaptive wait-free algorithms for lattice agreement and renaming. In *Proc. 17th ACM Symp. on Principles of Distr. Comp.* : 277-286, 1998.
12. H. Attiya and A. Fouren. Algorithms adaptive to point contention. In *J. ACM*, 50(4): 444-468, July 2003.
13. H. Attiya, A. Fouren and E. Gafni. An adaptive collect algorithm with applications. *Distributed Computing*, 15(2): 87-96, 2002.
14. H. Attiya, F. Kuhn, M. Wattenhofer and R. Wattenhofer. Efficient Adaptive Collect using Randomization. *Proc. 18th Conference on Distr. Comp. (DISC)*, 2004.
15. H. Attiya and I. Zach. Fully adaptive algorithms for atomic and immediate snapshots. WWW.CS.TECHNION.AC.IL/ HAGIT/PUBS/AZ03.PDF, 2003.
16. B. Englert and E. Gafni. Fast Collect in the Absence of Contention. *Proc. 22nd IEEE Intern. Conference on Distr. Comp. Systems (ICDCS)*: 537-543, 2002.
17. B. Englert and D. Goldstein. Can Memory be used adaptively by Uniform Algorithms? *Proc. 9th Intern. Conf. on Principles of Distr. Systems (OPODIS)*, 2005.
18. L. Lamport. A fast mutual exclusion algorithm. *ACM Transactions on Computer Systems*, 5(1): 1-11. February 1987.
19. M. Merritt and G. Taubenfeld. Speeding Lamport's fast mutual exclusion algorithm. *Information Processing Letters*, 45: 137-142, 1993.
20. M. Moir and J. Anderson. Wait-free algorithms for fast, long-lived renaming. *Science of Computer Programming*, 25(1): pp. 1-39, 1995.

On the Implementation of Parallel Shortest Path Algorithms on a Supercomputer*

Gabriele Di Stefano[1], Alberto Petricola[1], and Christos Zaroliagis[2]

[1] Dipartimento di Ingegneria Elettrica e dell'Informazione,
Università dell'Aquila, Italy
{gabriele, petricol}@ing.univaq.it
[2] Computer Technology Institute and University of Patras, Greece
zaro@ceid.upatras.gr

Abstract. We investigate the practical merits of a parallel priority queue through its use in the development of a fast and work-efficient parallel shortest path algorithm, originally designed for an EREW PRAM. Our study reveals that an efficient implementation on a real supercomputer requires considerable effort to reduce the communication performance (which in theory is assumed to take constant time). It turns out that the most crucial part of the implementation is the mapping of the logical processors to the physical processing nodes of the supercomputer. We achieve the requested efficient mapping through a new graph-theoretic result of independent interest: computing a Hamiltonian cycle on a directed hyper-torus. No such algorithm was known before for the case of directed hypertori. Our Hamiltonian cycle algorithm allows us to considerably improve the communication cost and thus the overall performance of our implementation.

1 Introduction

Computing shortest paths is one of the most fundamental problems in computer science and network optimization. Given an n-vertex, m-edge directed graph G with real edge costs, the *shortest path problem* asks for finding a path of minimum total cost from a vertex s to a vertex t, where the cost of a path is the sum of the costs of its edges. The *single-source shortest path* (SSSP) problem, which seeks for shortest paths from a specific vertex (source) s to all other vertices in G is a heavily studied problem. The most well-known algorithm for the case of non-negative edge costs is Dijkstra's algorithm [10] implemented with the help of an efficient priority queue; see e.g., [1]. A priority queue is a sequential data structure that maintains a set of elements with keys drawn from a totally ordered universe subject to the operations of insertion, deletion, decrease key, and find-minimum key element.

* This work was supported by the Consorzio Ricerche del Gran Sasso (CRGS), by the Human Potential Programme of EU under contract no. HPRN-CT-1999-00104 (project AMORE), and by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 406–417, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The efficient parallelization of a priority queue would immediately lead to the efficient parallelization of many sequential algorithms (including Dijkstra's algorithm for SSSP). For this reason, a significant thread of research has been devoted on designing efficient parallel priority queues. This research had been mainly organized along two directions. The first was to speed up the individual queue operations that handle a single element, using a small number of processors [3,6]. The second direction was to support the simultaneous insertion of k elements and the simultaneous deletion of the k smallest elements, k being a constant [6,9]. None of the above data structures supported the simultaneous deletion of k arbitrary elements.

An important step forward along the second direction was made in [4]. In that paper, a new parallel priority queue was presented that was the first to support the simultaneous insertion and simultaneous decrease key of an *arbitrary* sequence of elements ordered according to key, in addition to find-minimum and single element delete operations. Moreover, these operations can all be performed in $O(1)$ time. This allows for an efficient parallelization of Dijkstra's algorithm that runs in $O(n)$ time and $O(m \log n)$ work on an EREW PRAM. This algorithm constitutes one of the currently fastest and simultaneously work-efficient, deterministic, priority queue based, parallel algorithms for the SSSP problem.

The practicality of this data structure, however, had not been investigated before. The prime challenge for such an investigation is that the parallel algorithms in [4] have been designed to work on an EREW PRAM (the weakest version of the PRAM model; see [14,15]). PRAM is a shared memory model that abstracts several details of a real parallel machine, the most important of which concerns the (non realistic) assumption that shared memory allows a constant time direct communication between each pair of processors. Consequently, implementing a PRAM algorithm on a real parallel machine constitutes a great challenge, since processor communication is a bottleneck in performance.

Our main contribution in this work is an efficient implementation of the parallel priority queue in [4] and its practical assessment through the implementation and experimental evaluation of the parallel shortest path algorithm in the same paper on the APEmille supercomputer [2,17]. APEmille is a supercomputer whose processors are arranged as a three dimensional toroidal grid. Our study revealed that simulating the "all-to-all" processor communication in a straightforward manner incurs a considerable performance penalty. To alleviate this problem and provide an efficient implementation, we had to resort to the solution of a graph-theoretic problem; namely, to compute a Hamiltonian cycle in a directed hyper-torus. Our second contribution in this work is an algorithm, used to obtain a much better logical arrangement of processors, for computing such a Hamiltonian cycle in a directed hyper-torus, result of independent interest, since no such algorithm was known before for the case of directed hypertori.

Our implementation methodology provides also two results of independent interest: (a) the logical arrangement of processors of a toroidal grid along a Hamiltonian cycle turns out to be rather promising when the processors (of the ideal machine) have to work in a circular pipeline; (b) the number of transfers

required for a message from a logical (ideal) processor to reach its neighboring logical processor is at most 3. This improves upon the best previous result [12] that required 5 such transfers.

2 Computing a Hamiltonian Cycle on a Hyper-torus

Finding a Hamiltonian cycle in a hyper-torus (a Cartesian product of two or more cycles) is a well studied problem. It is known that the Cartesian product of any number of *undirected* cycles always contains a Hamiltonian cycle. Chen and Quimpo [7] obtained a stronger result, which provides a characterization of the pairs of vertices that can be joined by a Hamiltonian path, depending on the lengths of the cycles. In the case of *directed* cycles, Rankin [18] implicitly gave a necessary and sufficient condition for the existence of a Hamiltonian cycle in the Cartesian product of two directed cycles, and Trotter and Erdös [20] rediscovered later this characterization. Curran and Witte [8] showed that there is a Hamiltonian cycle in the Cartesian product of three or more nontrivial directed cycles. A sufficient condition for the existence of a Hamiltonian path in a torus has been recently given by Leontiev [16]. However, all these results do not provide a method to compute a Hamiltonian cycle.

Our main result in this section is an algorithm that computes a Hamiltonian cycle in a hyper-torus (under certain conditions) that runs in $O(\delta n)$ time. Proofs are omitted due to space limitations, but can be found in the full version [11].

Notations and definitions. The cartesian product $G = G1 \times G2$ of two directed graphs (digraphs) G_1 and G_2 is the digraph whose vertex set is $V(G) = V(G_1) \times V(G_2)$ and has an edge from (u_1, u_2) to (v_1, v_2) if and only if either $u_1 = v_1$ and there is an edge from u_2 to v_2 in G_2, or $u_2 = v_2$ and there is an edge from u_1 to v_1 in G_1. Let C_k denote a simple directed cycle with k vertices, each vertex numbered from 0 to $k-1$, and directed edges $(i, i+1 \mod k)$ for each $0 \leq i < k$. A *δ-dimensional directed toroidal grid* H is a graph obtained by the Cartesian product of δ cycles C_{d_i}, where $d_i \geq 2$ for each $i = 1, \ldots, \delta$. Then, a vertex x can be represented by a δ-tuple $(x_1, x_2, \ldots, x_\delta)$ of $\mathbb{Z}_{d_1} \times \mathbb{Z}_{d_2} \times \cdots \times \mathbb{Z}_{d_\delta}$. By definition of the Cartesian product of directed cycles, two adjacent vertices $u = (u_1, u_2, \ldots, u_\delta)$ and $v = (v_1, v_2, \ldots, v_\delta)$ differ only in one dimensional value, that is, $u_j \neq v_j$ for some $j \in \{1, \ldots, \delta\}$, and $u_i = v_i$ for each $i \neq j$. For two natural numbers a and b, we write $a \mid b$ if a divides b. Let $D_i = \prod_{j=0}^{i} d_j$, where $d_0 = 1$, and let $n = D_\delta$ be the number of vertices in H.

2.1 Hamiltonian Cycle

We start by providing a bijective function $f : \mathbb{Z}_n \longrightarrow \mathbb{Z}_{d_1} \times \cdots \times \mathbb{Z}_{d_\delta}$, which associates a natural number between 0 and $n-1$ to a vertex of H. Given some $t \in \mathbb{Z}_n$, the associated vertex $f(t) = (f_1(t), f_2(t), \ldots, f_\delta(t))$ is defined by:

$$f_i(t) = \begin{cases} \left\lfloor \frac{t}{D_{i-1}} \right\rfloor & i = \delta \\ \left(\left\lfloor \frac{t}{D_{i-1}} \right\rfloor - \sum_{j=i+1}^{\delta} f_j(t) \right) & \mod d_i \text{ else} \end{cases} \tag{1}$$

The reverse function f^{-1}, which associates a number in $\mathbb{Z}_n$ to a vertex $x = (x_1, x_2, \ldots, x_\delta)$, is given by:

$$f^{-1}(\ (x_1, x_2, \ldots, x_\delta)\ ) = \sum_{i=1}^{\delta} \left[\left(\sum_{j=i}^{\delta} x_j \right) \mod d_i \right] \cdot D_{i-1}$$

Lemma 1. *The function f is bijective.*

Theorem 1. *A δ-dimensional torus H obtained by the Cartesian product of δ directed cycles is Hamiltonian if $d_i \mid d_{i+1}$, for each $i = 1, \ldots, \delta - 1$.*

2.2 Labeling Algorithm

For each vertex of a δ-dimensional torus, we can use the function f to compute the correspondence between its coordinates and the ordering position in the Hamiltonian cycle. This implies a computational cost of $O(\delta^2 n)$. In the following, we present an algorithm working on a δ-dimensional torus that labels each node according to the function f, and which takes $O(\delta n)$ time. A formal description is given in Algorithm 1.

Algorithm 1. Node Labeling

Require: A δ-dimensional directed torus, s.t. d_i divides d_{i+1} , $1 \leq i \leq \delta - 1$.
Ensure: A labeling of vertices representing a Hamiltonian cycle.
1: Set $(x_1, x_2, \ldots, x_\delta) = (0, 0, \ldots, 0)$
2: Label vertex $(x_1, x_2, \ldots, x_\delta)$ with 0
3: **for** $t = 1, 2, \ldots, n-1$ **do**
4: Set $i = 1$
5: **while** $((x_1, x_2, \ldots, (x_i + 1) \mod d_i, x_{i+1}, \ldots, x_\delta)$ is labeled) **do**
6: Set $i = i + 1$
7: **end while**
8: Set $x_i = (x_i + 1) \mod d_i$
9: Label vertex $(x_1, x_2, \ldots, x_\delta)$ with t
10: **end for**

Theorem 2. *Algorithm 1 finds a Hamiltonian cycle in a* δ-dimensional directed torus *H obtained by the Cartesian product of δ cycles C_{d_i}, where $d_i \mid d_{i+1}$, for each $i = 1, \ldots, \delta - 1$. The algorithm takes $O(\delta n)$ time, where n is the number of vertices in H.*

In the case where the vertices of the torus do not know their absolute coordinates, Algorithm 1 can be executed to assign both labels and coordinates. It is sufficient each vertex to order its outcoming edges following the increasing size of the corresponding dimensions. A Hamiltonian cycle can be found in a distributed way starting from any vertex, labeled with 0, which initializes the algorithm. The task of the vertex is to interrogate in the correct order its neighbors, until it finds a non-labeled neighbor. This one is then labeled and it carries on the execution of the algorithm, following the same protocol.

3 The Parallel Priority Queue and the Shortest Path Algorithm

Dijkstra's algorithm [10] is a sequential algorithm for the *single-source shortest path problem* on directed graphs with non-negative real valued edge costs. Let $G = (V, E)$ be an n-vertex, m-edge directed graph with real-valued, non-negative edge costs $c(v, w)$, and let $s \in V$ be a distinguished source vertex. The single-source shortest path problem is to compute for all vertices $v \in V$ the length of a shortest path from s to v, where the length of a path is the sum of the costs of the edges on the path. Dijkstra's algorithm maintains for each vertex $v \in V$ a tentative distance $d(v)$ from the source and a set of vertices S for which a shortest path has been found. The algorithm iterates over the set of vertices of G, in each iteration selecting a vertex of minimum tentative distance which can be added to S. Priority queues are used in Dijkstra's algorithm for maintaining efficiently the tentative distances. One of the most efficient such queues is the Fibonacci heap [13], which results in the (asymptotically) fastest implementation of Dijkstra's algorithm that runs in $O(m + n \log n)$ time.

In the following we give a brief description of the Parallel-Dijkstra algorithm [4] designed to work on the EREW PRAM model of computation (see e.g., [15] for a discussion of PRAMs). The algorithm is based on the parallel implementation of a priority queue also presented in [4].

The priority queue maintains a set Q of elements e_i with a key d_i (we write $e_i(d_i)$ to indicate that an element e_i has key d_i). At any given instant, a set of successively numbered processors $P_1, \ldots, P_i$ will be associated with Q. The operations on the queue, in addition to INIT(Q) which initializes Q to the empty set and EMPTY(Q) which returns true if Q is empty, are:

- UPDATE(Q, L): updates Q with a list $L = e_1(d_1), \ldots, e_k(d_k)$ of (different) elements in non-decreasing key order, i.e., $d_1 \leq \ldots \leq d_k$. It allows (combined) multi-insert and multi-decrease key operations, i.e., if element e_i was not in the queue before the update, then e_i is inserted with key d_i; if e_i was already in Q with key d'_i, then the key of e_i is changed to d_i if $d_i < d'_i$. The sequence of elements must be given as a list, enabling one processor to execute in constant time the access operations.
- DELETEMIN(Q): deletes and returns the minimum key element from Q.
- DELETE(Q, e): deletes element e from Q.

These operations are executed by the processors associated with the queue in parallel. The UPDATE operation, as an additional task, assigns a new processor to Q.

A simple parallel implementation of the priority queue can be by obtained organizing the processors associated with Q in a linear pipeline: when an UPDATE(Q, L) operation is performed, the new processor associated with Q is put at the front of the pipeline.

All operations are based on the procedure MERGESTEP(Q) [4]: let P_i denote the ith processor to become associated with Q and L_i the element list associated

with P_i. The MergeStep(Q) performs, in parallel on each processor, one step of a merge of the element list L_i of P_i and the elements in the output queue Q_{i-1} of the previous processor, which is a standard FIFO queue.

The elements in each output queue will be in non-decreasing order, so the merge step of processor P_i simply consists in taking the smaller element of either Q_{i-1} or L_i if neither is empty; if either Q_{i-1} or L_i is empty the element is taken from the other sequence, and when both are empty, P_i has no more work to do.

In order to guarantee that an element is output at most once (implementing the multi-decrease), each processor maintains a Boolean array F_i of forbidden elements ($F_i[e] =$ **true** iff e has been chosen to output and made forbidden by P_i) and the MergeStep(Q) procedure maintains the invariants that $F_i \cap Q_{i-1} = \emptyset$ and $F_i \cap L_i = \emptyset$. To this end, each processor maintains also two arrays of pointers $\overline{Q}_i$ and $\overline{L}_i$ into Q_i and L_i, respectively, indexed by the elements.

Let us consider, for example, the sequential pipeline shown in Figure 1: P_1, P_2, and P_3 select, respectively, 5(15), 2(12), and 1(10) to output and mark 5, 2, and 1 as forbidden; 2(12) is output to Q_2 by P_2 and 1(19) is output to Q_3 by P_3, while 5(15) is not output to Q_1 by P_1 because $5 \in F_2$ (to have $F_2 \cap Q_1 = \emptyset$); P_2 removes 2(14) from L_2 too (to have $F_2 \cap L_2 = \emptyset$).

	L_i	Q_i	F_i
P_1	5(15) 4(17) 7(19)	2(12) 1(14)	1 2 3
P_2	4(13) 2(14)	5(11)	3 5
P_3	1(10) 5(14) 4(18) 2(19)		
P_1	4(17) 7(19)	1(14)	1 2 3 5
P_2	4(13)	5(11) 2(12)	2 3 5
P_3	5(14) 4(18) 2(19)	1(10)	1
P_4	4(13) 6(15)		

Fig. 1. A sequential pipeline before and after Update(Q, 4(13) 6(15))

A formal description of the Parallel-Dijkstra algorithm that uses the parallel priority queue is given in Algorithm 2. The notation $L_v(d)$ means that when vertex v is selected, the list object L_v of its adjacent edges will be initialized

Algorithm 2. Parallel-Dijkstra

Require: A graph $G = (V, E)$ with non-negative real edge costs and a source vertex $s \in V$.
Ensure: Single source shortest path
1: /* Initialization */
2: For each $v \in V$ sort the adjacency lists L_v of G after edge cost;
3: For each $v \in V$ build array I_v of vertices w which v is adjacent to ($(w, v) \in E$);
4: Init(Q);
5: $d(s) \leftarrow 0$; $S \leftarrow \{s\}$;
6: Update(Q, $L_s(0)$);
7: /* Main loop */
8: **while** $\neg$Empty(Q) **do**
9: $v(d) \leftarrow$ DeleteMin(Q);
10: $d(v) \leftarrow d$; $S \leftarrow S \cup \{v\}$;
11: Update(Q, $L_v(d)$);
12: **for all** $w \in I_v$ **par do**
13: **if** $w \notin S$ **then** remove v from L_w;
14: **end for**
15: **end while**

with a constant value d, representing the distance of v from the source, that will be an offset value to be added to each key of the elements of L_v.

Lines 9 to 11 of the main loop represent the classic Dijkstra's steps, in which the parallel priority queue is exploited for finding the vertex of minimum distance and decreasing the distances of its adjacent vertices. An extra work, performed by lines 12 to 14, is required to prevent that a vertex, once selected and added to the set S for which a shortest path has already been found, is ever selected again. The problem is that when vertex v is selected by the find minimum operation, some of its adjacent vertices may have been selected at a previous iteration. Such vertices have to be removed from the adjacency list of v and this is done by using I_v, the array of vertices w to which v is adjacent (i.e., $(w, v) \in E$).

4 Implementation on the APEmille Supercomputer

4.1 The APEmille Architecture

APEmille [2,17] is a massively parallel supercomputer in the Teraflop range whose architecture was originally optimized for the simulation of some major Physics problems. APEmille is an array of Processing Nodes operating in SIMD (Single Instruction Multiple Data) mode [19]. Each node (Jmille) is based on a pipelined floating-point arithmetic processor (500 MFlops) with its own locally addressable data memory (32 MB). Each group of eight nodes is controlled by a Control Processor (Tmille) with its own data memory (512 KB) and a separate program memory (40 MB). Nodes are logically arranged as a three dimensional toroidal grid, and connected to first neighbors by a synchronous network of Communication Processors (Cmille), which supports homogeneous communications, i.e., all nodes access simultaneously data from a corresponding remote node with a given relative distance. Programming in APEmille is carried out in TAOmille [5], a high level language with a syntax very similar to Fortran, but augmented with various language constructs to facilitate parallel programming. For our work we used an APEmille with 128 nodes, 64Gflops and 8GB RAM.

4.2 A First Implementation

The described linear pipeline version of the Parallel-Dijkstra algorithm has been implemented on the APEmille supercomputer. To obtain the best possible performance of the implementation, a set of optimizations has been added to the code, some of which are of a general character, while some others are pertinent to the APEmille architecture.

Data have been logically distributed in a way that is different from the original algorithm, maintaining its structure unaltered, but limiting the need for a remote data transfer to only one couple element-key $e(d)$ for each MergeStep(Q) call.

In the remainder of the exposition, we will distinguish between physical processors (the APEmille processing nodes or PNs) and logical processors P_i used in the algorithm. To actually parallelize the algorithm on the real machine, we have to assign a logical processor to a physical node.

For each UPDATE operation, the algorithm requires a new logical processor to be activated and linked to the processor activated in the previous step. The activation of the physical nodes is done in an order imposed by the algorithm's evolution. As a consequence, it can be fairly possible that two consecutive logical processors might not correspond to two adjacent nodes in the grid (especially if there is no particular strategy for associating logical processors to PNs).

Homogeneous communications do not allow, in this case, a simultaneous data transferring between pairs of nodes. In order to reduce the number of communications, in a first version of the implementation, we developed and used an operation whose cost is proportional to the sum of the dimension sizes of the grid. The idea is to move data along the three directions. Each node prepares its message ($e(d)$) that is transferred along the x dimension performing $x - 1$ communications between adjacent nodes. After this phase, each node receives all messages sent by nodes with the same x coordinate. By repeating this activity for the y and z dimension, each message will reach every node in the grid (see Figure 2) and then its destination.

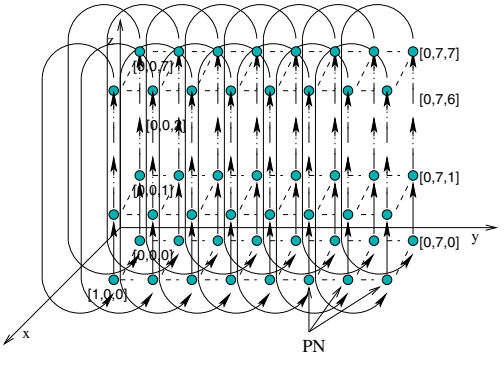


Fig. 2. Message transfer along z direction: after the message transfers along the x and the y dimension have been completed, each node has all the messages sent by nodes in the same x-y level. In $z - 1$ steps, each node will receive all the sent messages.

4.3 Improved Implementation

Our initial experimental findings (see Section 5) showed that simulating the "all-to-all" processor communication as described in the previous section incurs a considerable performance penalty. A better logical arrangement of processors is required in order to reduce the communication cost. To this end, we used Algorithm 1 (detailed in Section 2).

Algorithm 1 produces a Hamiltonian cycle in a hyper-torus if d_i divides d_{i+1}, $1 \leq i \leq \delta - 1$. Since APEmille is a toroidal grid in which each dimension has a number of nodes which is a power of 2, the above constraints are satisfied. Hence, Algorithm 1 can be used to find a cyclic path (i.e., a sequence of distinct linked nodes) in the grid that visits all nodes of the architecture. Figure 3 shows the resulting cycle for a $2 \times 8 \times 8$ APEmille configuration.

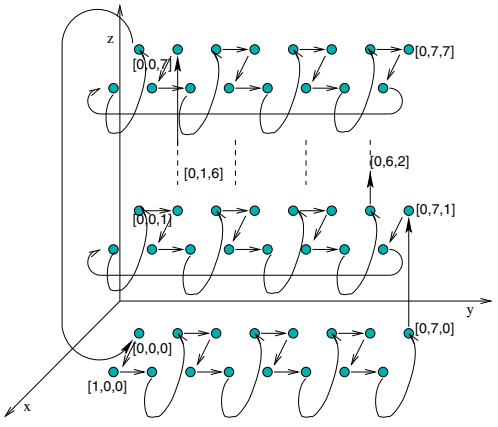


Fig. 3. Hamiltonian cycle for a $2 \times 8 \times 8$ APEmille configuration

Using the ordering of the nodes along the Hamiltonian cycle, we associate each logical processor P_i with the i-th PN in the Hamiltonian cycle ordering. An important consequence is that now two logically consecutive processors are associated to two adjacent PNs. In this way, the source of a message can reach its destination is a single homogeneous transfer. Since such a destination can be in three possible directions, three homogeneous transfers are sufficient to reach the next logical processor. This is an improvement upon the result in [12], where an algorithm is proposed to determine a Hamiltonian cycle in the APEmille tori that requires five transfers between adjacent nodes. The above results in the second version of our implementation, which drastically reduces the number of communications.

5 Experimental Evaluation

All performance measurements were made on different APEmille configurations with 8, 32 and 128 PNs. As we are interested in analyzing the behavior of the Parallel-Dijkstra algorithm on large graphs, we implemented it in such a way that a single PN manages $\lceil n/N \rceil$ vertices, where n is the number of vertices and N the number of PNs.

A set of different graphs on n vertices has been randomly generated with a number of edges proportional to $O(n \log n)$, $O(n^{1.5})$, and $O(n^2/\log n)$, respectively. Although due to memory restrictions and available node configurations we were not able to run experiments for values of n larger than 1500 (with a few exceptions), these values allow us to draw conclusions about the algorithm's performance. Edge weights are non-negative reals chosen randomly from $[0, 10000]$.

Table 1 shows the results of the first version (Section 4.2) of our implementation of the Parallel-Dijkstra algorithm. For each experiment the following results are reported: the number n of nodes and the number m of edges; the ratio n/N indicating the number of vertices managed by a PN, and the execution time (in seconds) for 8, 32, 128 PN APEmille configurations.

Table 1. Performance of the first version

		8 nodes		32 nodes		128 nodes	
vertices	edges	n/N	time (s)	n/N	time (s)	n/N	time (s)
500	3100	63	2.45	16	0.90	4	0.40
500	11130	63	2.45	16	0.90	4	0.40
500	40751	63	2.45	16	0.90	4	0.40
1000	6900	126	9.80	32	3.60	8	1.60
1000	31456	126	9.80	32	3.60	8	1.60
1000	139540	126	9.80	32	3.60	8	1.60
1500	11060	188	22.05	47	8.10	12	3.60
1500	58742	188	22.05	47	8.10	12	3.60
1500	306652	188	22.05	47	8.10	12	3.60

The main result is that the execution time of the algorithm does not depend on the number of edges, as expected. However, the execution time does not reduce proportionally to the number of processors. This is due to the cost of the communication operation, which increases with the number of nodes.

Table 2 shows the results of the second (improved) version of our implementation of the Parallel-Dijkstra algorithm. The experiments are performed on the same graphs and for each experiment the same set of results is reported.

Table 2. Performance of the second (improved) version

		8 nodes		32 nodes		128 nodes	
vertices	edges	n/N(avg)	time (s)	n/N(avg)	time (s)	n/N(avg)	time (s)
500	3100	11.7	0.36	3.6	0.12	1.6	0.06
500	11130	15.2	0.49	4.5	0.17	1.7	0.10
500	40751	16.3	0.64	4.6	0.29	1.7	0.21
1000	6900	23.4	1.40	6.6	0.42	2.3	0.17
1000	31456	30.4	1.90	8.3	0.61	2.7	0.28
1000	139540	31.8	2.40	8.5	1.08	2.7	0.71
1500	11060	35.2	3.15	9.5	0.90	3.1	0.33
1500	58742	45.6	4.23	12.1	1.30	3.6	0.56
1500	306652	47.5	5.35	12.4	2.28	3.6	1.50

Compared to Table 1, the execution times reduce drastically. The improvement is mainly due to the optimized communication step, described in Section 4.3. As in the previous case, the execution time does not reduce proportionally to the number of nodes, even if the time needed to perform the communications is independent from the machine configuration. This is due to the work made by the lines 12 to 14 of the Parallel-Dijkstra algorithm. Actually, in the second version the association of a logical processor with a graph vertex is done during the execution of the Parallel-Dijkstra algorithm, and it is not possible to determine a priori which vertex will be managed by a particular PN. The operations have therefore to be executed by all the "not yet activated" PNs for each non-visited vertex w adjacent to v, leading to a linear time cost proportional to $|I_v|$.

Another reason that contributes to the performance improvement is the variation of the average number of vertices managed by a single node. This value has been computed as the sum of vertices managed by a node for every step divided by the number of steps. It is not constant, because the ordered activation

solution allows for an additional optimization in the code. In fact, during the execution, there is a progressive emptying of the structures L and Q. When the data structures are empty, the processor is deactivated. As consecutive blocks of logical processors are managed in parallel, and as the emptying follows the order of the logical processors, the number of operations can be reduced considering the progressive deactivation of such blocks, and performing the algorithm operations only on active blocks.

Finally, we would like to compare our parallel implementation with the sequential Dijkstra's algorithm. For that purpose, we have implemented the sequential algorithm on APEmille and executed it on a single processor. The sequential algorithm uses Fibonacci heaps [13] as its priority queue.

Table 3 reports the experimental results obtained on the previous set of graphs. As expected, the execution time grows with the number of vertices and with the connectivity of the graph.

Table 3. Performance of sequential Dijkstra with Fibonacci heaps

vertices	edges	time (s)
500	3100	0.16
500	11130	0.20
500	40751	0.37
1000	6900	0.34
1000	31456	0.51
1000	139540	1.06
1500	11060	0.57
1500	58742	0.85
1500	306652	2.08

Comparing the performances of both parallel and sequential versions of the algorithm (see Tables 1, 2 and 3), we observe the following.

The first implementation of our algorithm is always worse than the sequential one on the graph sizes we considered and on any configuration of the APEmille. It is worth mentioning that our first implementation managed to beat the sequential algorithm on a synthetic graph instance with 5000 vertices and 6000000 edges on a 128-node APEmille configuration. Clearly, on larger graph instances as well as on larger APEmille configurations, our first implementation is expected to perform better than the sequential algorithm.

Our second (improved) implementation compares favorably with the sequential algorithm on the 32-node APEmille configuration, and it is clearly faster than the sequential algorithm on the 128-node APEmille configuration. Consequently, a better speedup is expected on larger graph instances and/or larger APEmille configurations.

References

1. Ahuja, R.K., Magnanti, T.L., Orlin, J.B.: Network Flows. Prentice-Hall (1993)
2. Aglietti, F. et al.: An overview of the APEmille parallel computer. Nucl. Inst. And Methods in Phys. **A 389** (1997) 56–58

3. Brodal, G.: Priority queues on parallel machines. Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT), LNCS, **1097** (1996) 416–427
4. Brodal, G., Träff, J., Zaroliagis, C.: A Parallel Priority Queue with Constant Time Operations. Journal of Parallel and Distributed Computing **49** (1998) 4–21
5. Cabasino, S., D'Autilia, R., Paolucci, P.S., Todesco, G.M.: The TAO language. http://www-zeuthen.desy.de/ape/html/APEmille/Documentation/ (1995)
6. Chen, D.Z., Hu, X.: Fast and efficient operations on parallel priority queues. Algorithms and Computation: 5th Int. Symp. (ISAAC93), LNCS, **834** (1994) 279–287
7. Chen, C.C., Quimpo, N.F.: On strongly Hamiltonian abelian group graphs. Lecture Notes in Mathematics, **884** (1981) 23–34
8. Curran, S.J., Witte, D.: Hamilton paths in Cartesian products of directed cycles. Ann. Discrete Mathematics, **27** (1985) 35–74
9. Das, S.K., Pinotti, M.C., Sarkar, F.: Optimal and load balanced mapping of parallel priority queues in hypercubes. Trans. Par. Dist. Syst. **7** (1996) 555–564 [Corr. 896]
10. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numer. Math. **1** (1959) 269–271
11. Di Stefano, G., Petricola, A., Zaroliagis, C.: On the Implementation of Parallel Shortest Path Algorithms on a Supercomputer. Tech. Rep. Univ. of L'Aquila (2006)
12. Francia, M., Panizzi, E., Petricola, A., Visconti, G.: Parallel Simulation of Orography Influence on Large-Scale Atmosphere Motion on APEmille. ACM Computing Frontiers '04 (2004) 320–325
13. Fredman, M.L., Tarjan, R.E.: Fibonacci Heaps and their uses in Improved Network Optimization Algorithm. Journal of ACM **34** No.3 (1987) 596–615
14. Jájá, J.: An Introduction to Parallel Algorithms. Addison-Wesley, Reading (1992)
15. Karp, R.M., Ramachandran, V.: Parallel Algorithms for Shared-Memory Machines. Handbook of Theoretical Computer Science **A**, C. 17, Elsevier (1990) 869–942
16. Leontiev, V.K.: Hamiltonian cycles in torical lattices. DMTCS Conference **AE** (2005) 397-400
17. Panizzi, E., Sacco, G.: The APEmille project. LNCS, HPCN2000 (2000) 539–542
18. Rankin, R.A.: A campanological problem in group theory. Proc. Camb. Phil. Soc. **44** (1948) 17–25
19. Stone, H.S.: High Performance Computer Architecture. Addison-Wesley, (1990)
20. Trotter, W., Erdös, P.: When the Cartesian product of directed cycles is Hamiltonian. J. Graph Theory **2** (1978) 137–142

An Efficient K-Coverage Eligibility Algorithm on Sensor Networks

Meng-Chun Wueng and Shyh-In Hwang

Department of Computer Science and Engineering
Yuan Ze University
Chungli, Taiwan, R.O.C.
{jun, shyhin}@mmlab.cse.yzu.edu.tw

Abstract. Wireless sensor networks are employed in many critical applications. The K-coverage configuration is usually adopted to guarantee the quality of surveillance. A sensor node can be determined to be *ineligible* to stay active when its sensing range is K-covered. Although many algorithms have been proposed to reduce the complexity of the K-coverage configuration, the accuracy cannot be preserved when the number of deployed sensor nodes increases. In this paper, we propose an efficient K-coverage eligibility (EKE) algorithm to accurately and cheaply determine the eligibility of each sensor node. The algorithm focuses on the regions having a lower degree of coverage for each sensor node. Therefore, the complexity of the EKE algorithm is reduced substantially while retaining accuracy. Experimental studies indicated that the computational cost of the EKE algorithm could be reduced by up to 89% and that the correct percentage was larger than 90%.

1 Introduction

Advances in micro-sensor and wireless communication technologies have led to small and inexpensive sensor nodes that perform cooperative tasks for important applications, including surveillance, target tracking, military tasks, and hazardous environment exploration [1]. Because sensor nodes may exhibit faulty behavior, related fault-tolerant technologies are investigated to guarantee the quality of applications on sensor networks. The faulty behavior of sensor nodes may result from many factors, such as a faulty decision from the signal processing in a senor node because of noise [2], environmental interference, or battery depletion, or through malfunctions arising from low-cost hardware. From a study of a system comprising 70 MICA2 sensor nodes, He et al. [3] noted that faulty sensor nodes increase power consumption unnecessarily and lead to unpredictable results. Clouqueur et al. [4] attempted to reduce the number of Byzantine faults of sensor nodes by employing value fusion and decision fusion schemes. Because an individual sensor node may not be reliable, a higher degree of coverage is necessary to mask the faults of sensor nodes and obtain a higher confidence in detection. Therefore, many coverage-preserving scheduling schemes are proposed to guarantee the required coverage degree while minimizing the number of active sensor nodes [5,6,3,7,8,9].

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 418–429, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Most coverage-preserving scheduling schemes divide the operation into rounds. Each round begins with a self-organizing phase followed by a sensing phase. For the self-organizing phase, many K-coverage configuration algorithms have been proposed to guarantee that each location in the monitored area will be covered by at least K active sensor nodes [8,10,11,9]. Because the operation of maintaining the K-coverage configuration is executed periodically or frequently, it is important to reduce the cost of the operation for long-term monitoring on sensor networks.

Given a required K-coverage degree, a fundamental problem is how to determine that the monitored area is K-covered. Wang et al. [9] proved that this problem could be transformed to calculate the coverage degree of each sensor node within the monitored area. Furthermore, the coverage degree of each node can be obtained by tracing, within the sensing range, all points that are intersected by its neighbors. To reduce the power consumption, the authors proposed the use of a K-coverage eligibility (KE) algorithm. A sensor node can be determined to be *ineligible* to stay active if all of the intersection points within its sensing range are already K-covered by its neighbors. Therefore, the number of active sensor nodes can be reduced while still guaranteeing the surveillance quality. Although the deterministic K-coverage eligibility algorithm can accurately determine the eligibility of each sensor node, the computational cost is $O(n^3)$ where n is the number of the neighbors within twice the sensing range of each node.

Different from the above deterministic algorithm, Huang and Tseng [11] calculated the coverage degree of each sensor node by tracing its perimeter segments covered by its neighbors, called K-perimeter-covered (KPC) algorithm. A sensor node can be turned off if its perimeter is covered by at least K neighbors. This algorithm effectively reduced the complexity to $O(n \log n)$. However, since the algorithm only focuses on the coverage on the perimeter of a node, the coverage within the sensing range cannot be known. According to our experimental results, the accuracy of the algorithm is averagely 82% when 100 sensor nodes area uniformly deployed in a 50m×50m monitored area and the required K-coverage degree is 1. When the number of deployed sensor nodes increases, the accuracy cannot be guaranteed.

In this paper, we propose an efficient K-coverage eligibility (EKE) algorithm that can correctly determine the eligibility of each sensor node with low cost. A distinct feature of the EKE algorithm is that the neighbors of each sensor node are classified into R_neighbors and 2R_neighbors, which are defined in Section 3. Instead of calculating the coverage degree of all intersection points within the sensing range of a node, the EKE algorithm only requires to focus on the candidate intersection points surrounding the lower coverage regions based on the characteristics of the R_neighbors and 2R_neighbors. Therefore, the computational cost of the EKE algorithm can be highly reduced. Because the algorithm aims, however, to determine the regions that exhibit a lower coverage degree and not the minimal coverage degree, its accuracy may be diminished somewhat. Although the accuracy of the EKE algorithm cannot be guaranteed

as 100%, according to the experimental results, the correct percentage is larger than 90% as the number of the deployed sensor nodes increases. Furthermore, the computational cost is only 11% of that of the deterministic algorithm proposed by Wang et al. [9]. Because wireless sensor networks have scarce energy resource, it is acceptable to have less than 10% of the locations in the monitored area under K-coverage [10,12,9]. For some critical applications, fault tolerant technologies can be further exploited to guarantee the quality of the applications. Therefore, we argue that the proposed EKE algorithm is beneficial to preserve the surveillance quality in a long-term monitoring task on sensor networks.

In the next section, we briefly review the related work in the literature. Section 3 presents the design issues and inaccurate cases. Simulation results are presented and discussed in Section 4. We present our conclusions in Section 5.

2 Related Work

Many coverage-preserving configurations and node-scheduling algorithms have been investigated in an effort to guarantee the surveillance quality of applications on sensor networks [5,6,11,9]. Wang et al. [9] proved that if a patch having a lower degree of coverage is inside the sensing range of a node and is surrounded by the node's neighbors or the monitored edges, then the patch and the intersection points that bind it will have the same coverage degree. Therefore, the coverage degree of the patch can be obtained by calculating the degree of the intersection points. If each intersection point within the sensing range is already K-covered by its neighbors, the sensor node is ineligible to be active. The KE algorithm can determine the eligibility of sensor nodes accurately, but the computational cost is quite high. To reduce the energy consumption without losing the accuracy of this operation, we adopted the idea proposed by Wang et al., but we trace only some of the candidate intersection points, rather than all of the intersection points within twice the sensing range of each node.

Huang and Tseng [11] have also attempted to reduce the computational cost of the K-coverage configuration. They proposed a K-perimeter-covered (KPC) algorithm to calculate the coverage degree of each sensor node by tracing the perimeter segments covered by its neighbors. Because this algorithm does not require the coverage to be considered within the sensing range of a node, the computational cost can be effectively reduced, but at the expense of reduced accuracy of determining the eligibility for each sensor node. Furthermore, the KPC algorithm ignores the fact that the sensor nodes located near the monitored edges have some invalid perimeter segments, even though the coverage degree of invalid perimeter segments should not be calculated. Therefore, a sensor node located near the monitored edges may be determined to be eligible to become active, but its sensing range within the monitored edge is already K-covered in reality. With increasingly more sensor nodes are deployed, the accuracy of the KPC algorithm will decrease even further.

3 Efficient K-Coverage Eligibility Algorithm

In this section, we present the efficient K-coverage Eligibility (EKE) algorithm. Before describing the algorithm, several assumptions and definitions are presented. The design considerations and the inaccurate cases of the algorithm are interpreted as follows.

3.1 Formal Definitions of the Algorithm

In this research, all sensor nodes with identical sensing range, R, are assumed to be location-aware, and no other sensor node locates at the same position in the monitored area. For calculating the coverage degree of a sensor node, we define that an arbitrary point p is covered by a sensor node s if their Euclidian distance is less than the sensing range s, that is, $d(i, j) < R$. With the physical consideration of signal decay of a sensor node, a point which is located exactly at the sensing range of a sensor may not be detected correctly. Therefore, it is reasonable to assume that p is not covered by s if $d(i, j) = R$.

Based on the above assumptions, the coverage degree of a sensor node depends on how the neighbors within *2R* cover it. Similar to the KE algorithm [9] and the KPC algorithm [11], each node in our algorithm needs to collect the neighbor information within *2R*. However, we do not determine the eligibility of a sensor node by calculating the coverage degrees of all neighbor intersection points inside the sensing range of a sensor node or by tracing the perimeter segments covered by all neighbors on a sensor node. Here we classify the neighbor set of each node into two groups, called R_neighbors and 2R_neighbors.

Definition 1. *R_neighbors and 2R_neighbors. The R_neighbors of a sensor node* i *is defined as* $R\_neighbors(i) = \{j \mid j \in N,\ j \neq i,\ d(i, j) < R\}$ *where* N *is the set of sensor nodes located in the monitored area, and* d(i,j) *denotes the distance between sensor node* i *and sensor node* j. *The 2R_neighbors of sensor* i *is defined as* $2R\_neighbors(i) = \{j \mid j \in N,\ j \neq i,\ R \leq d(i, j) < 2R\}$.

3.2 Design Issues of the Algorithm

There are two reasons why we classify the neighbor set of a sensor node into two groups and calculate their coverage degree individually. The first notion is that while farther from the target sensor node *i*, 2R_neighbors tend to form an area with lower coverage degree inside the sensing range of the node, as depicted in Figure 1(a). Even if 2R_neighbors are very close to the sensor node, as shown in Figure 1(b), the node will not be fully covered by all 2R_neighbors based on the assumption that a point is not covered when it is located exactly at the sensing range of a node. Hence, when a sensor node has only 2R_neighbors, the coverage degree of the node can be determined immediately, that is 1. The second notion is that the number of R_neighbors is bounded by the sensing range of a node. Moreover, in many cases even if a sensor node has R_neighbors and 2R_neighbors, the eligibility of the node can be determined by only tracing the

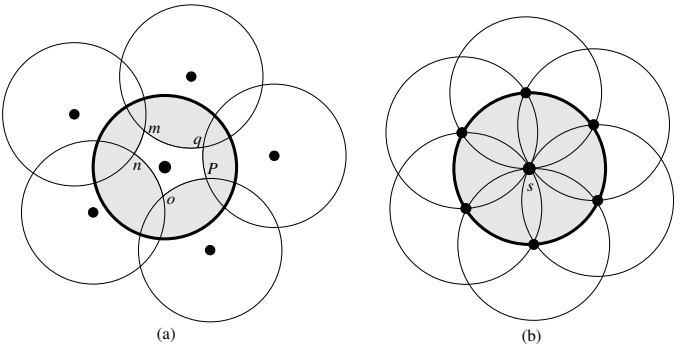


Fig. 1. (a) The central area surrounded by m, n, o, p, q has lower coverage degree. (b) The coverage degree of sensor node s is equal to 1.

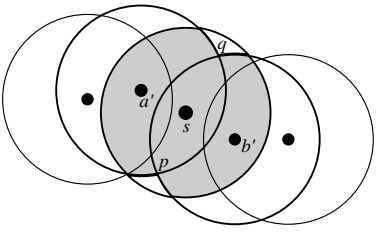


Fig. 2. The eligibility of node s can be determined by tracing only p and q

intersection points of R_neighbors, as illustrated in Figure 2. Therefore, if we can classify the neighbors of a sensor node into two groups, and calculate the coverage degree of the points intersected by R_neighbors first, the computational cost in many cases can be bounded by the number of R_neighbors. Although the number of R_neighbors will increases as more and more sensor nodes are deployed, the number of R_neighbors is 1/3 of 2R_neighbors when the number of the deployed sensor nodes is large enough.

The eligibilities of many sensor nodes can be determined after tracing the intersection points of their R_neighbors. As the number of the deployed sensor node increases, the coverage within the sensing range of each node is complicated. The methods of calculating the coverage degree of sensor nodes can be classified into three cases. The first case is that a sensor node has only one R_neighbor and several 2R_neighbors, as depicted in Figure 3(a). The second one is that a sensor node is located near the edge of the monitored area, so that the intersection points of R_neighbors inside the sensing range of the node may fall out of the monitored area, as p in Figure 3(b). The third case is that the intersection points with the minimal coverage degree of R_neighbors are covered by some 2R_neighbors, as i is covered by a and b in Figure 4(a). The above three cases mean that the algorithm needs to discover other lower coverage regions inside the sensing range of the node.

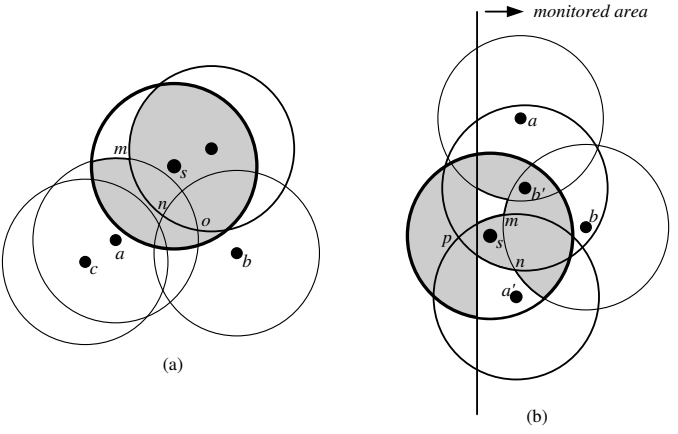


Fig. 3. (a)A sensor node has only one R_neighbor. (b)The intersection points of R_neighbors fall out of the sensing range of the node and the monitored area.

For the first case, since one R_neighbor cannot cover the entire sensing range of a sensor node, and the 2R_neighbors form a lower coverage region in the center of the node, the lower coverage regions will be surrounded by the R_neighbors and other 2R_neighbors. The regions can be discovered by finding out the points with the minimal coverage degree intersected by the R_neighbor and the 2R_neighbors, as m intersected by a' and a in Figure 3(a). Even if the R_neighbor does not have any intersection with the 2R_neighbors, the minimal coverage is the sensor node itself. The eligibility of the node can still be determined. For the second case, although the intersection points of R_neighbors cannot be used to determine the eligibility of the node, in most cases, the eligibility of the node can still be decided by tracing the points intersected by any two R_neighbors and 2R_neighbors, as m and n in Figure 3(b). During the processing, the algorithm terminates when the coverage degree of any intersection point is less than the required K-coverage degree. In this algorithm, we do not trace the points intersected by any two 2R_neighbors. Since the operation not only incurs lots of computations, but also cannot find out the intersection points with lower coverage degree quickly.

The third case is more complicated and requires further explanation. To clearly explain the case, we introduce several keywords. The intersection points of any two R_neighbors covered by the fewest R_neighbors are called *candidate intersection points* and the two R_neighbors are called *candidate R_neighbors*. The 2R_neighbors that cover the *candidate intersection points* are represented as *candidate 2R_neighbors*. Besides, the *decision points* mean the points intersected by the *candidate R_neighbors* and the *candidate 2R_neighbors*.

In this case, the *candidate intersection points* are covered by several *candidate 2R_neighbors*. The lower coverage regions usually can be found by tracing the *decision points* and the *candidate intersection points*. Take Figure 4(a) as an example, the intersection point i is the *candidate intersection point* which is covered by the *candidate 2R_neighbors*, a and b. The lower coverage region is

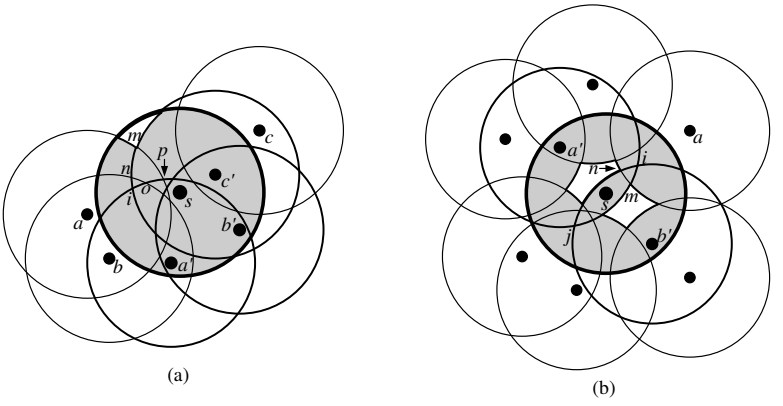


Fig. 4. Two cases of the lower coverage region formed by the candidate neighbors

surrounded by m which has the minimal coverage degree among m, n, o, p, q, i which are intersected by *a'*, *c'*, *a*, and *b*. In this way, the coverage of the 2R_neighbors within the sensing range of the R_neighbors will just increase the coverage degree of the sensor node, as the coverage of c in Figure 4(a). The coverage degree in this overlap is usually higher than the required K-coverage degree. Therefore, to determine the eligibility of a node with low computational cost, after the lower coverage regions surrounded by the R_neighbors are found, we focus that how the 2R_neighbors cover the founded regions. If the *candidate 2R_neighbors* do not fully cover the region, as Figure 4(a), a new lower coverage region will be formed by the *candidate R_neighbors* and the *candidate 2R_neighbors*. Therefore, we only need to trace their intersection points and find out the points with the minimal coverage degree. On the other hand, if the *candidate 2R_neighbors* fully cover the regions, since the coverage of 2R_neighbors on the sensing range of a node is limited, the lower coverage regions will be surrounded in the center by the *candidate R_neighbors* and the *candidate 2R_neighbors*, as the regions surrounded by m and n in Figure 4(b). With the consideration that there may have complicated intersections in the center of the node and the *candidate intersection points* has the minimal coverage degree. Therefore, the algorithm will trace the coverage degree of the *decision points* and the *candidate intersection points*.

3.3 Inaccurate Cases

Although the algorithm is workable in most cases, various coverages of the R_neighbors and the 2R_neighbors of a sensor node may still cause the eligibility of a few sensor nodes unable to be accurately determined. The inaccurate cases can be classified into two types. One type is caused by the monitored edges. For the effect of the monitored edges, this algorithm only considers whether the *candidate intersection points* are outside the edges or not. Therefore, when the lower coverage is surrounded by the points intersected by the R_neighbors and

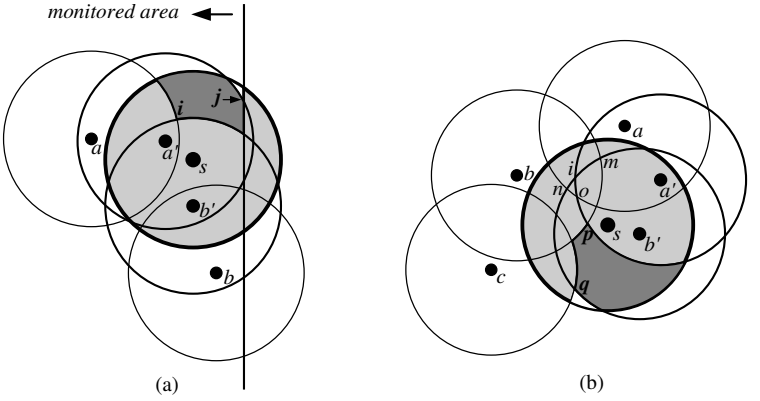


Fig. 5. Two inaccurate cases

the monitored edge, the eligibility of the node cannot be accurately decided. Take Figure 5(a) as an example, the algorithm will determine the eligibility of the node depending on the coverage degree of i not j. Actually, this case can be resolved by further tracing the points of the R_neighbors and the 2R_neighbors intersected by the monitored edges. Since less than half of the inaccurate cases caused by monitored edges, to reduce the computational cost, the algorithm does not implement the processing. The other error case is caused by various deployments of the R_neighbors and the 2R_neighbors of a sensor node. Figure 5(b) belongs to the case that the *candidate intersection points* are covered by several *candidate 2R_neighbors*, but the eligibility of the node cannot be determined by only tracing the *decision points* and the *candidate intersection points* when the required K-coverage degree is 1. The real decision point in Figure 5(b) is q not p. For this case, there is no a fast and correct rule can be used to predict the lower coverage regions, except tracing each point of any two neighbors. This is a tradeoff between the accuracy of the algorithm and the computational cost.

4 Performance Evaluation

In this section, we evaluate the performance of the EKE algorithm on NS-2 in terms of accuracy and computational cost. Three related algorithms are also implemented to compare with the EKE algorithm, including the K-coverage eligibility (KE) algorithm [9], the K-perimeter-covered (KPC) algorithm [11], and the Grid algorithm. The simulation environment is a 50m×50m square space, and the sensing range of all deployed sensor nodes is 5m. Each result is the average of five runs with different random network topologies. All algorithms terminate when the coverage degree of a sensor node is less than or equal to the required K-coverage degree.

Figure 6 illustrate the accuracy and the number of processing times of the four algorithms when the required K-coverage degree is 2, i.e. K=2. The KE algorithm can precisely determine the eligibility of each sensor node. Even if a

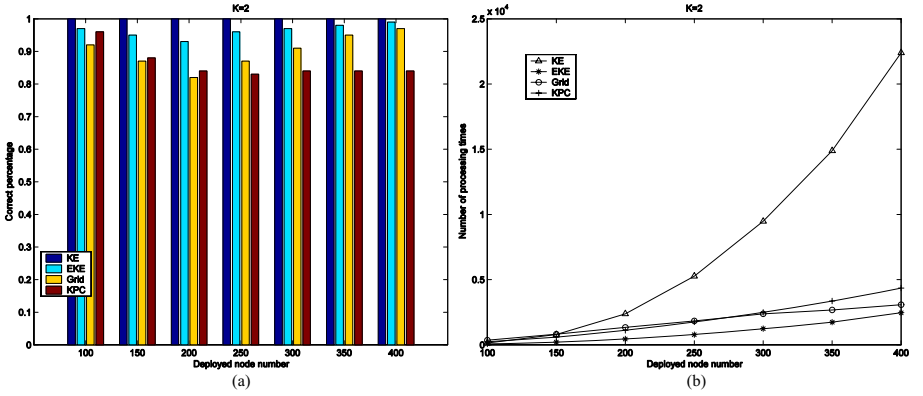


Fig. 6. (a)The correct percentages of the four algorithms. (b)The number of processing times of the four algorithms.

sensor node is located near the monitored edges, its eligibility can also be decided by tracing the points intersected by the edges and its neighbors within the sensing range. However, the computational cost of the KE algorithm is considerably high as the deployed sensor nodes increase. Although the KPC algorithm effectively reduces the complexity of K-coverage configuration, its accuracy cannot be guaranteed. This is because the coverage degree in the center of the sensing range cannot be obtained by only tracing the perimeter segments of each sensor node. Furthermore, the KPC algorithm does not distinguish the sensor nodes located near the monitored edges from those fully inside the monitored area. The perimeter segments outside the monitored area are also calculated. Therefore, many sensor nodes are considered to have lower coverage degrees and lots of redundant nodes need to be active. When there are more and more sensor nodes deployed, the number of nodes near the monitored edges also increases, so that the error ratio of the algorithm cannot be reduced.

Similar to the goal of the KPC algorithm, the simple Grid algorithm is usually used to approximately determine the coverage degree of a monitored area. In our evaluation, the sensing area of each sensor node is divided into 1m×1m grids. The coverage degree of each grid is obtained by calculating how many active sensor nodes cover the center of the grid. The eligibility of each sensor node can thus be determined by tracing all grids within its sensing range. In this algorithm, the major issue is how to determine the grid size, because both the accuracy and the computational cost are affected by the size. Figure 6(b) shows that when the number of the deployed nodes is less than 150, the computational cost of the scheme is higher than the other three algorithms. However, as the number of the deployed nodes increases, its computational cost can be bounded by the number of the grids.

Compared to the performance of the KPC and Grid algorithms, the proposed EKE algorithm has the highest correct ratio and the lowest computational cost, as illustrated in Figures 6. This is because the EKE algorithm classifies the neighbors of each sensor node into R_neighbors and 2R_neighbors. Based on the

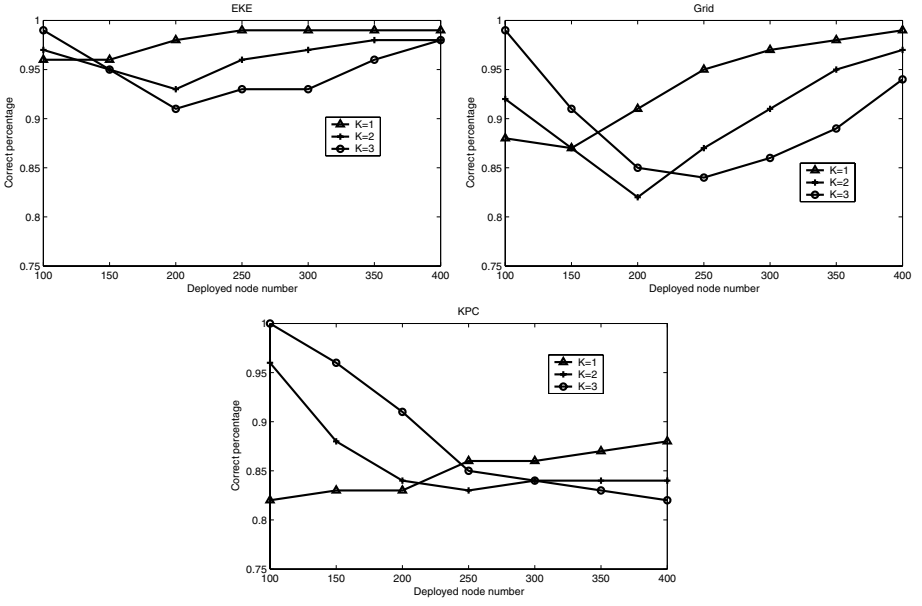


Fig. 7. The correct percentages of the EKE, Grid, and KPC algorithms

characteristics of the two groups, the eligibility of each node can be determined by only tracing the intersection points surrounding the lower degree regions rather than all intersection points within its sensing range.

Figure 7 presents the correct percentages of the EKE, Grid, and KPC algorithms under different required K-coverage degrees. When K=1, the accuracy of all algorithms is improved as more and more sensor nodes are deployed. This is because when the number of the deployed nodes is large enough, all sensor nodes in the monitored area will have the coverage degree higher than the required one. Therefore, all algorithms can correctly determine the coverage degree of most nodes. In the KPC algorithm, since the number of the redundancy error case is also increased, the improvement of its correct percentage is limited. On the other hand, as the required K-coverage degree increases, if the coverage degrees of most sensor nodes are less than the required one, the accuracy of all algorithms can be guaranteed. However, when there are more and more nodes deployed, the regions covered by all neighbors on a sensor node is more complicated than the node has fewer neighbors. Therefore, the probability of accurately determining the eligibility of sensor nodes by the three algorithms is thus reduced. Similarly, when the number of the deployed nodes is large enough, even if all algorithms cannot find out the minimal coverage degree of sensor nodes, the eligibility of many sensor nodes can be determined correctly. In the proposed EKE algorithm, the correct percentage can be guaranteed to be more than 90% in all cases. In Figure 8, as the required K-coverage degree increases, the computational costs of the KE, EKE, KPC, and Grid algorithms are all reduced. This is because all algorithms terminate when the coverage degree of a sensor node is less than or

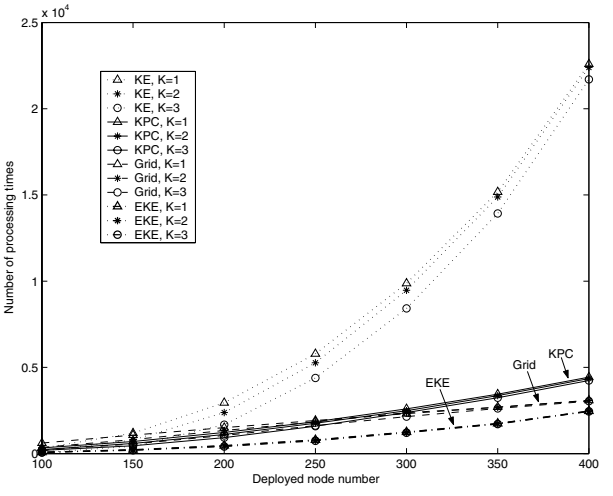


Fig. 8. The computational cost of the four algorithms under different required K-coverage degrees

equal to the required degree. Since the proposed EKE algorithm focuses on tracing the regions with lower coverage degree in all sensor nodes, the computational cost is always the lowest among all algorithms.

5 Conclusions and Future Work

In this paper, we proposed an efficient K-coverage eligibility (EKE) algorithm, which is able to accurately determine the eligibility of sensor nodes at low cost. We presented that the neighbors of each sensor node can be classified into R_neighbors and 2R_neighbors. Based on the characteristics of the two groups, the lower coverage regions of each sensor node can be discovered. Therefore, only some candidate intersection points surrounding the lower coverage regions, rather than all intersection points of each sensor node, need to be examined. Simulation results demonstrate that the accuracy of the EKE algorithm is higher than 90% and the computational cost is only 11% of a deterministic algorithm. Furthermore, the proposed algorithm has the highest correct percentage and the lowest computational cost among all other approximate algorithms.

Although the EKE algorithm effectively guarantees the K-coverage configuration with low complexity, how to design an efficient node-scheduling algorithm to preserve the coverage is still left open and will be investigated in the future. Since the EKE algorithm sometimes fails to determine the eligibilities of a few sensor nodes, how to employ a data aggregation scheme to guarantee the surveillance quality of critical applications on sensor networks will also be studied.

References

1. Ilyas, M., Mahgoub, I.: Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems. CRC (2004)
2. Varshney, P.K.: Distribution Detection and Data Fusion. Springer-Verlag (1996)
3. He, T., Krishnamurthy, S., Stankovic, J.A., Abdelzaher, T., Luo, L., Stoleru, R., Yan, T., Gu, L.: Energy-Efficient Surveillance System Using Wireless Sensor Networks. In: Proc. of the ACM International Conference on Mobile Systems, Applications, and Services (MobiSys). (2004) 270–283
4. Clouqueur, T., nd P. Ramanathan, K.K.S.: Fault Tolerance in Collaborative Sensor Networks for Target Detection. IEEE Transactions on Computers **53** (2004) 320–333
5. Tian, D., Georganas, N.K.: A Coverage-Preserving Node Scheduling Scheme for Large Wireless Sensor Networks. In: Proc. of the ACM International Workshop on Wireless Sensor Networks and Applications (WSNA). (2002) 32–41
6. Yan, T., He, T., Stankovic, J.A.: Differentiated Surveillance for Sensor Networks. In: Proc. of the ACM International Conference on Embedded Networked Sensor Systems (SenSys). (2003) 51–62
7. Hsin, C.F., Liu, M.: Network Coverage Using Low Duty-Cycled Sensors: Random & Coordinated Sleep Algorithms. In: Proc. of the ACM International Symposium on Information Processing in Sensor Networks (IPSN). (2004) 433–442
8. Kumar, S., Lai, T.H., Balogh, J.: On K-coverage in a Mostly Sleeping Sensor Network. In: Proc. of the ACM International Conference on Mobile Computing and Networking (MobiCom). (2004) 144–158
9. Xing, G., Wang, X., Zhang, Y., Lu, C., Pless, R., Gill, C.: Integrated Coverage and Connectivity Configuration for Energy Conservation in Sensor Networks. ACM Transactions on Sensor Networks **1** (2005) 36–72
10. Zhang, H., Hou, J.: On Deriving the Upper Bound of α-Lifetime for Large Sensor Networks. In: Proc. of the ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc). (2004) 121–132
11. Huang, C.F., Tseng, Y.C.: The Coverage Problem in a Wireless Sensor Network. Mobile Networks and Applications **10** (2005) 519–528
12. Bai, H., Chen, X., Ho, Y.C., Guan, X.: Percentage Coverage Configuration in Wireless Sensor Networks. In: Proc. of the International Symposium on Parallel and Distributed Processing and Applications (ISPA), LNCS 3758. (2005) 780–791

A Distributed Algorithm for a b-Coloring of a Graph

Brice Effantin and Hamamache Kheddouci

Laboratoire PRISMa, Université Lyon 1,
IUT A Département Informatique,
01000 Bourg-en-Bresse, France
{beffanti, hkheddou}@bat710.univ-lyon1.fr
http://www710.univ-lyon1.fr/~g2ap/

Abstract. A b-coloring of a graph is a proper coloring where each color admits at least one node (called *dominating node*) adjacent to every other used color. Such a coloring gives a partitioning of the graph in clusters for which every cluster has a clusterhead (the dominating node) adjacent to each other cluster. Such a decomposition is very interesting for large distributed systems, networks,... In this paper we present a distributed algorithm to compute a b-coloring of a graph, and we propose an application for the routing in networks to illustrate our algorithm.

1 Introduction

In a distributed system, a node exchanges information only with its neighborhood. Every node has a set of local variables to determine a local state of the node. The state of the entire system, called *global state*, is the union of the local states of all the nodes in the system. The objective in a distributed system is to obtain automatically a desirable global final state (called *legitimate state*) where each node is considered as a distinct entity able to compute its own state itself, with its neighborhood as only knowledge. Distributed algorithms are a very attractive topic for a lot of fields and several graph problems arise naturally in distributed systems. For example, distributed algorithms for finding spanning trees, matchings, independent sets or particular colorings have been studied [1,5,9,15,16]. Such algorithms are so very interesting and efficient for dynamic networks [3,17].

The aim of our paper is to propose a distributed method to determine a particular coloring of graphs. Thus, we propose a partitioning method under conditions of a graph based on a graph coloring called *b-coloring*. A *proper k-coloring* is a coloring using k colors such that two adjacent nodes have different colors. Then, a *b-coloring* is a proper k-coloring where for each color i, $1 \leq i \leq k$, there exists a node x, with color i, adjacent to nodes colored with every color j, $1 \leq j \neq i \leq k$. Such nodes are called *dominating nodes*. This coloring was introduced by Irving and Manlove in [10] where they presented the *b-chromatic number* as the maximum integer k such that G admits a b-coloring with k colors. In [10], they also proved that finding the b-chromatic number of any graph is

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 430–438, 2006.
© Springer-Verlag Berlin Heidelberg 2006

a NP-hard problem and they gave a polynomial-time algorithm for finding the b-chromatic number of trees. This parameter was also studied for other classes of graphs like cartesian product of graphs [13], bipartite graphs [14], power graphs of paths and cycles [7]. More recently, Corteel et *al.* [6] proved that the b-chromatic number problem is not approximable within $120/133-\epsilon$ for any $\epsilon > 0$, unless $P = NP$. This coloring is very interesting since it partitions the nodes of G into color classes for which at least one node is adjacent to each other color class. The property of these nodes is very attractive to give a hierarchy of nodes, to exchange information with other classes,...

The remainder of this article is then decomposed as follows. In Section 2, we present a distributed algorithm to compute a b-coloring of a graph. Then in Section 3, we propose an application to this clustering where we study a routing method. Finally, Section 4 concludes with future works.

We first start with some definitions used in the following. A distributed system will be modeled with an undirected connected graph $G = (V, E)$, where the node set V represents the set of processors, and the edge set E represents the processor interconnexions. Throughout this paper, we assume that $|V| = n$ and $|E| = m$. For every node i, we define $N(i)$, its *open neighborhood*, as the set of nodes adjacent to i (the set of colors of $N(i)$ will be denoted $N_c(i)$). We let $d(i) = |N(i)|$, the number of neighbors of node i, or its *degree*, and we let $\Delta = \max\{d(i)|i \in V\}$. Finally, let $diam(G)$ be the diameter of the graph G, defined as the maximum distance between any pair of vertices of G. In our study we assume a synchronized model in which any process computes the same action at the same time. Then, two or more processes can be in conflict (for example, two adjacent processes colored with the same color). A central daemon selects, among all these processes, the next process to compute. Thus, if two or more processes are in conflict, we cannot predict which process will be computed next. Several protocols exist ([2,4]) that provide a scheduler. Our algorithm can be combined with any of these protocols to work under different schedulers as well.

It is very important to remark that the coloring constraint in this problem is not only on the neighborhood of each node, but on the entire graph. Indeed, the color of a node depends on the presence or not of a dominating node for its color. Note that in the following we say that a color is *emptyable* if it has no dominating nodes. Thus, the first idea for finding such a coloring is: 1- Find a proper coloring of the graph; 2- For any emptyable color c, recolor nodes with color c by other colors and discard color c. However, the b-coloring is not a continue coloring. For instance, the hypercube Q_3 admits b-colorings with 2 or 4 colors but not with 3 (see Figure 1). Thus if we remove emptyable colors, we must remove them one by one and verify, at each step, if the coloring is a b-coloring or not.

2 b-Coloring of a Graph

In this section we propose a distributed algorithm to determine a partitioning (or a clustering) of the nodes of a graph G based on a particular coloring. In

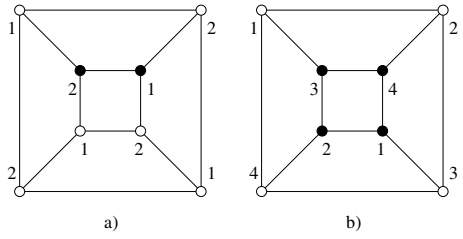


Fig. 1. b-colorings with k colors for the hypercube Q_3, where $a)$ $k = 2$, $b)$ $k = 4$. (black nodes are dominating nodes).

every cluster we select a clusterhead (called the dominating node in the coloring) able to join directly each other cluster.

In this approach, the nodes are represented by processes and they search to have a color with dominating node (*i.e.* the emptyable colors are removed from the coloring). Since the information on colors are dispersed in the entire graph, nodes need to exchange some messages with the remainder of the graph. Our approach enables the nodes to react to these messages to reach a legitimate state for the system. Thus nodes have different labels from the interval $[1; n]$. The principle of this algorithm is to detect emptyable colors. If such colors exist, they must be removed, but as we saw, one by one.

This method is a recoloring method. So nodes must be initialized. To put the system in an initialized state, we color a node with maximum degree by color 1 and we use Procedure 1.

Procedure 1. *Init_Coloring()*
begin
if $c(i) \neq \emptyset$ then
  Let $M = N_c(i) \cup \{c(i)\}$.
  $q = 0$.
  for every node $j \in N(i)$ such that $c(j) = \emptyset$ do
    $q = \min\{k | k > q, k \notin M$ and $k \notin N_c(j)\}$.
    if $q \leq \Delta + 1$ then $c(j) = q$ else $c(j) = \min\{k | k \notin N_c(j)\}$. endif.
  enddo.
endif.
end.

Lemma 1. *Procedure 1 is done in $O(m)$.*

Proof. Procedure 1 is applied once on every colored node. For each node, the procedure colors every of its non-colored neighbors. Since each node has $d(i)$ neighbors, Procedure 1 is computed in $\sum_{i=1}^{n} d(i) = 2m = O(m)$. □

Several studies propose fast colorings of graphs which can be used to initialized the state of nodes. In particular Hedetniemi et *al.* in [8], and Johansson in [12], propose distributed colorings using at most $\Delta + 1$ colors. Our procedure has the advantage to find exactly $\Delta + 1$ colors in a linear time.

To compute a b-coloring of a graph, any node needs some information about dominating nodes and emptyable colors. We let for any node i, $Dom_i[]$ as a table containing the label of the dominating node for each color (initialized to 0). If a color c has no dominating node we have $Dom_i[c] = 0$ and the value $Dom_i[c] = \emptyset$ means that the color c has been removed from the coloring. Then, before removing a color c, every node must verify if it is not a dominating node for this color. If it is not, it will send a message to propose to remove the color c.

Before the study of each possible message, we give the two following procedures. The first evaluates the state of a node (dominating node or not). By comparing the set of existing colors in the graph and the set of colors in its neighborhood, a node determines if it is a dominating node or not. Then, between all possible nodes to be a dominating node for a color c, we choose the node with the smallest label. If i is not a dominating node and if there does not exist a dominating node for $c(i)$, then node i supposes that its color is an emptyable color. Note that each node maintains also the color list of its neighborhood $N_c(i)$.

Procedure 2. *Processing*()
begin
Let $N'_c = \bigcup q$ such that $1 \leq q \neq c(i) \leq \Delta + 1$ and $Dom_i[q] \neq \emptyset$.
if $N'_c = N_c(i)$ (*i.e.* i is a dominating node) then
  if $Dom_i[c(i)] > i$ or $Dom_i[c(i)] = 0$ then
    $Dom_i[c(i)] := i$.
    send to every $k \in N(i)$: $M_2(i, c(i))$.
  endif.
else
  if $Dom_i[c(i)] = 0$ then
    send to every $k \in N(i)$: $M_3(c(i))$.
    Execute $Waiting()$.
  endif.
endif.
end.

The second procedure gives some instructions computed after a delay, used to consider that a message reached every node in the graph. This procedure will be applied on a node i if it determines its color $c(i)$ as emptyable. Since the method is distributed, any node works in the same time. Thus, if there exists a dominating node j for $c(i)$, this information will be propagated to every node of G in time $diam(G)$. We will see that if the node i receive this information, it stops the execution of the Procedure 3. So, if i waits for a delay $diam(G) + 1$, it can consider that its color has no dominating node, and it will replace it.

Procedure 3. $Waiting()$
begin
After a time $diam(G) + 1$ do
  $col := c(i)$.
  $c(i) := \max\{q | 1 \leq q \leq \Delta + 1,\ q \notin N_c(i)$ and $Dom_i[q] \neq \emptyset\}$.
  send to every $k \in N(i)$: $M_1(i, c(i))$.

send to every $k \in N(i)$: $M_4(col)$.
end.

Then, every node i will react following the received message. Messages are presented below:

- No message is received. In this case node i applies Procedure 2 to evaluate its state.
- Message $M_1(j, c)$: "NODE j HAS COLOR c."
  Every node receiving this message updates the colors of its neighborhood.
- Message $M_2(j, c)$: "NODE j IS A DOMINATING NODE FOR THE COLOR c."
  Among all nodes verifying the conditions to be a dominating node for color c, the dominating node of c will be the node with the smallest label. Then, if j is a dominating node for $c(i)$ (*i.e.* $c(i) = c$), then node i interrupts its procedure $Waiting()$. Moreover, to limit the number of messages, the node i propagates this information only if it is new.
  Then, if this message is received, node i applies the following procedure:

  Procedure 4. $M_2(j, c)$
  begin
  if $Dom_i[c] > j$ or $Dom_i[c] = 0$ then
  $Dom_i[c] := j$.
  send to every $k \in N(i)$: $M_2(j, c)$.
  if $c = c(i)$ then Interrupt $Waiting()$. endif.
  endif.
  end.

- Message $M_3(c)$: "COLOR c IS EMPTYABLE."
  Among all emptyable colors, the next color to remove will be the smallest. This message is so used by any node to determine if its color is the next color to remove. Thus, if the color received by node i is smaller than $c(i)$, then it propagates the message. Moreover, if i executes currently the procedure $Waiting()$ (Procedure 3), then it stops it since the next color to remove will not be $c(i)$.

  Procedure 5. $M_3(c)$
  begin
  if $c < c(i)$ and $Dom_i[c] = 0$ then
  send to every $k \in N(i)$: $M_3(c)$.
  Interrupt $Waiting()$.
  endif.
  end.

- Message $M_4(c)$: "DELETE COLOR c"
  If $c(i) = c$ then the node i must take another existing color and it propagates this message to remove color c from the graph. Moreover, since the color of node i perhaps changed, it stops its procedure $Waiting()$ and starts again the procedure $Processing()$.

Procedure 6. $M_4(c)$
begin
if $c(i) = c$ then
$Dom_i[c] := \emptyset$.
$c(i) := \max\{q | 1 \leq q \leq \Delta + 1,\ q \notin N_c(i)$ and $Dom_i[q] \neq \emptyset\}$.
send to every $k \in N(i)$: $M_1(i, c(i))$.
endif.
send to every $k \in N(i)$: $M_4(c)$.
Interrupt $Waiting()$.
Execute $Processing()$.
end.

Proposition 1. *A legitimate state for G is reached with $O(n\Delta)$ local changes and $O(m\Delta^2)$ exchanged messages.*

Proof. First, a color is removed if it is emptyable, and only one color is removed at the same time. Thus, each node can change at most Δ times its color. Hence the total number of local changes is $O(n\Delta)$.

Then, we can evaluate the number of exchanged messages. If the color of a node changes, this node sends its new color to its $d(i)$ neighbors. Since a node can change at most Δ times its color, we have $\sum_{i=1}^{n} \Delta.d(i) = O(m\Delta)$ Messages 1 sent. For Message 2, suppose that every node is a dominating node. Then, a node can receive at most $n - 1$ messages on dominating nodes, and it can also transmit these messages to its $d(i)$ neighbors. Thus $\sum_{i=1}^{n}(n-1).d(i) = O(nm)$ messages are used to propagate these information. For the emptyable colors (Message 3), a node sends a message to its neighborhood every time it finds a smaller color to remove. Suppose that every color is emptyable. Since at most Δ colors can be removed, each node sends at most $\Delta.d(i)$ messages. Every time a color is removed, the same reasoning can be done. Thus, we deduce that $\sum_{i=1}^{n}(\sum_{q=1}^{\Delta} q.d(i)) = O(m\Delta^2)$ Messages 3 are exchanged. Finally, by the same way as for Message 2, we deduce that $O(nm)$ messages are used to remove colors (Message 4). Therefore, the number of exchanged messages to join a legitimate state is in $O(m\Delta^2)$. □

Theorem 1. *The time complexity is $O(\Delta diam(G))$.*

Proof. After the initialization step, any node determines dominating nodes in time $diam(G)$ (by the propagation of Message 2). During the same time, an emptyable color can be found (if exists). However, in the worst case, two opposite information can traverse the graph and prevent us from determining an emptyable color (since any node interrupted its Procedure 3). Nevertheless, in this case no more messages are exchanged, and from the algorithm every node applies Procedure 2. Thus, in time $diam(G)$, an emptyable color is found (if exists).

Consequently, an emptyable color starts to be removed after a delay $diam(G) + 1$ and it is completely removed in time $diam(G)$. Since at most Δ colors are removed, a legitimate state is computed in $O(\Delta.diam(G))$. □

Figure 2 presents an example of computation of the algorithm.

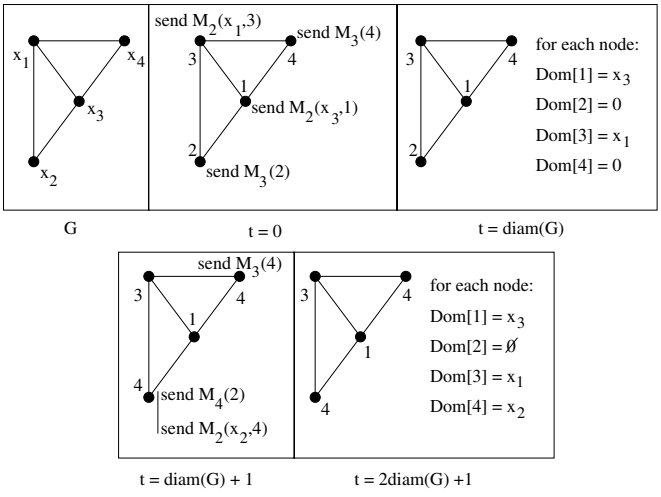


Fig. 2. Messages:$M_2(x, c)$ means "x is a dominating node for color c";$M_3(c)$ means "c is an emptyable color";$M_4(c)$ means "delete color c". Thus, at $t = 0$ the graph G is initialized and each node x_i evaluates its state (Procedure 2). At $t = diam(G)$, every node knows the dominating nodes and the next color to remove. At $t = diam(G) + 1$, Procedure 3 on node x_2 is terminated, so it changes its color. At $t = 2diam(G) + 1$, nodes know the dominating nodes and the coloring is found.

3 A b-Coloring Based Routing Method

In this section we propose an application of the above algorithm, for routing information between nodes of a weighted graph. The idea is to decompose the graph in clusters, and to use the clusterhead of the source node to join the cluster of the target node.

To adapt our algorithm to a routing application, it needs some minor modifications. Indeed, the clusterhead of each cluster will communicate with the nodes of its cluster. We just add some data to compute theses paths. Thus, we define for each node i, $Predec_i[c]$ as the predecessor of i in the path from the dominating node of color c and i, and $Dist_i[c]$ as the length (*i.e.* the total weight) of this path. Thus, Procedure 2 initializes these data for any dominated color found. Then, Message 2 can include two new parameters: the node v from which the message comes, and its distance to the dominating node j. Then, the node i can modify its path and so its distance to j in function of these parameters. These modifications will be integrated to the messages propagating the domination of j. This computation can increase the number of Message 2 sent but does not change the complexity of the algorithm. Finally, information on the path and the distance about a removed color can be destroyed in Message 4.

These modifications of the algorithm presented in the previous section enables us to compute the shortest path from a clusterhead to any node of its cluster in the same time as the b-coloring of the graph. Then, to propagate information

from a source node s to a target node t, we use the dominating nodes of the clusters containing s and t. Let w_s and w_t be the dominating nodes of clusters containing respectively s and t. Thus, the steps for determining a path from s to t are:

1. Finding the path from s to w_s,
2. Joining a neighbor w' of w_s in the cluster containing node t (property of the clusterhead w_s),
3. Finding the path from w' to w_t,
4. Finding the path from w_t to t.

The Figure 3 presents the method used to determine a route between a source s and a destination t.

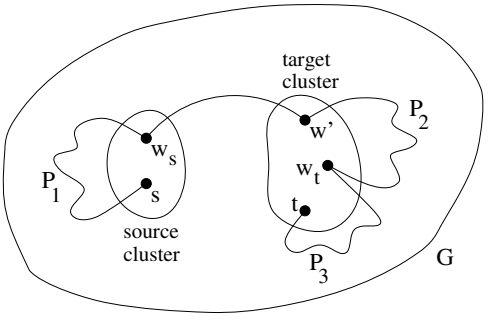


Fig. 3. Illustration of the routing from the source node s to the target node t in the graph G. Any path P_i is computed in the same time as the coloring and every path P_i is a shortest path between its endvertices.

Lemma 2. *Routing method complexity is* $O(n)$.

Proof. For the determination of a path from a clusterhead x to a member of its cluster y, we need to traverse the graph from y to x, node by node. Indeed in the algorithm, each node knows only its predecessor in the path from x to y. Thus, a path between any node and its clusterhead is done in $O(n)$ (this is the case of steps $1, 3$ and 4). Moreover, each node knows its neighborhood. So, the step 2 is done in $O(1)$. Therefore, a path from a source node to a destination node in a graph G is computed in $O(n)$. □

This method depends on the complexity of the b-coloring problem. It does not improve existing methods based on spanning trees [1,3,5] or routing table [11], it just proposes a new way to route information by a clustering of the structure.

4 Conclusion

In this article, we presented a distributed algorithm to partition a structure in clusters. Each of these clusters has a clusterhead with the property to join

directly every other cluster. An interesting question completes this study: How can react vertices if the topology of the graph evolves ? Our study concerns currently the dynamic graphs.

References

1. Antonoiu, G., Srimani, P.K.: A self-stabilizing distributed algorithm for minimal spanning tree problem in a symmetric graph. Computer & Mathematics with Application **35**(10) (1998) 15-23
2. Antonoiu, G., Srimani, P.K.: Mutual exclusion between neighboring nodes in an arbitrary system graph tree that stabilizes using read/write atomicity. Proceedings Euro-par'99, LNCS **1685** (1999) 823-830
3. Baala, H., Flauzac, O., Gaber, J., Bui, M., El-Ghazawi, T.: A self-stabilizing distributed algorithm for spanning tree construction in wireless ad hoc networks. Journal of Parallel and Distributed Computing **63** (2003) 97-104
4. Beauquier, J., Datta, A.K., Gradinariu, M., Magniette, F.: Self-stabilizing local mutual exclusion and daemon refinement. Proceedings DISC 2000, LNCS **1914** (2000) 223-237
5. Bui, M., Butelle, F., Lavault, C.: A distributed algorithm for constructing a minimum diameter spanning tree. Journal of Parallel and Distributed Computing **64** (2004) 571-577
6. Corteel, S., Valencia-Pabon, M., Vera, J.-C.: On approximating the b-chromatic number. Discrete Applied Mathematics **146** (2005) 106-110
7. Effantin, B., Kheddouci, H.: The b-chromatic number of some power graphs. Discrete Mathematics and Theoretical Computer Science **6** (2003) 45-54
8. Hedetniemi, S.T., Jacobs, D.P., Srimani, P.K.: Linear time self-stabilizing colorings. Information Processing Letters **87** (2003) 251-255
9. Hsi, S-C., Huang, S-T.: A self-stabilizing algorithm for maximal matching. Information Processing Letters **43**(Issue 2) (1992) 77-81
10. Irving, R.W., Manlove, D.F.: The b-chromatic number of a graph. Discrete Applied Mathematics **91** (1999) 127-141
11. Ito, H., Iwama, K., Okabe, Y., Yoshihiro, T.: Single backup table schemes for shortest-path routing. Theoretical Computer Science **333** (2005) 347-353
12. Johansson, Ö.: Simple distributed $\Delta+1$-coloring of graphs. Information Processing Letters **70** (1999) 229-232
13. Kouider, M., Mahéo, M.: Some bounds for the b-chromatic number of a graph. Discrete Mathematics **256**(Issues 1-2) (2002) 267-277
14. Kratochvíl, J., Tuza, Z., Voigt, M.: On the b-chromatic number of graphs. 28th International Workshop on Graph-Theoretic Concepts in Computer Science, Cesky Krumlov, Czech Republic; LNCS **2573** (2002) 310-320
15. Shi, Z., Goddard W., Hedetniemi, S.T.: An anonymous self-stabilizing algorithm for 1-maximal independent set in trees. Information Processing Letters **91** (2004) 77-83
16. Šparl, P., Žerovnik, J.: 2-local distributed algorithms for generalized coloring of hexagonal graphs. Electronic Notes in Discrete Mathematics **22** (2005) 321-325
17. Srivastava, S., Ghosh, R.K.: Distributed algorithms for finding and Maintaining a k-tree core in a dynamic network. Information Processing Letters **88** (2003) 187-194

Topology-Sensitive Epidemic Algorithm for Information Spreading in Large-Scale Systems

J. Acosta-Elías, J.M. Luna-Rivera, M. Recio-Lara, O. Gutiérrez-Navarro, and B. Pineda-Reyes

Facultad de Ciencias de la Universidad Autónoma de San Luis Potosí
Av. Salvador Nava s/n, Zona Universitaria
San Luis Potosí, S.L.P., 78290, México
Tel.: +52 (444) 826 2316, Fax: +52(444) 826 2321
jacosta@uaslp.mx, mlr@fciencias.uaslp.mx, mrecio@galia.fc.uaslp.mx, ognavarro@galia.fc.uaslp.mx, bpineda@uaslp.mx

Abstract. Epidemic algorithms are an emerging technique that has recently gained popularity as a potentially effective solution for disseminating information in large-scale network systems. For some application scenarios, efficient and reliable data dissemination to all or a group of nodes in the network is necessary to provide with the communication services within the system. These studies may have a large impact in communication networks where epidemic-like protocols become a practice for message delivery, collaborative peer-to-peer applications, distributed database systems, routing in Mobile Ad Hoc networks, etc. In this paper we present, through various simulations, that an epidemic spreading process can be highly influenced by the network topology. We also provide a comparative performance analysis of some global parameters performance such as network diameter and degree of connectivity. Based on this analysis, we propose a new epidemic strategy that takes into account the topological structure in the network. The results show that the proposed epidemic algorithm outperform a classical timestamped anti-entropy epidemic algorithm in terms of the number of sessions required to reach a consistent state in the network system.

1 Introduction

The widespread introduction of networking in modern society has provoked a spectacular progress in the areas of large-scale computer and communication networks. The proliferation of large-scale networks has facilitated to reduce the emphasis on groups of people at work and in the community and afforded a turn to networked societies that are loosely bounded. Asynchronous collaborative applications in large-scale networks are complex systems consisting of many components whose operation depends on many processes. The Internet is an example of a large-scale, massively-distributed, and highly-interacting communication network of devices (i.e. computers) characterized by explosive growth, extreme heterogeneity, and unpredictable or even chaotic dynamic behavior. A major challenge in these networks is the development of reliable algorithms to

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 439–450, 2006.
© Springer-Verlag Berlin Heidelberg 2006

disseminate information from a given source (node) to thousands or millions of users (rest of the nodes in the network).

The provision of communication services within a network system is essential to allow an entity to send information toward another entity or to broadcast it to all the entities of the system. However, this process is not an easy task in a large-scale and distributed network which changes arbitrarily in a self organizing way. Since the number of nodes in such networks can be quite huge and the environment dynamic, i.e., nodes can die or move, then its topology can change constantly. In the last few years, epidemic algorithms have been proposed as a more efficient and reliable data dissemination technique for this type of network systems [1,2]. Epidemic algorithms, in which each entity (node) propagates the information by sending it to a sub-set of its neighborhood, are network protocols that allow rapid dissemination of information from a source through purely local interactions. This data dissemination process represents a mechanism analogous to the disease propagation in populations or the spread of rumor in social networks [3,4]. Therefore, the distributed and decentralized nature of epidemic algorithms implies that these techniques potentially find a large spectrum of application in computer and communication networks such as mobile communication networks [5], peer to peer communication networks [6,7], grid based networks [8], etc.

The goal of this paper is to investigate new epidemic data dissemination strategies which can exploit the topological properties of large-scale distributed networks. In this work, we first explore the performance behavior, in terms of the message propagation time (anti-entropy sessions), of a conventional timestamped anti-entropy epidemic algorithm [9] through several simple network topologies. From these results, it is shown that performance of epidemic algorithms is highly influenced by the network topology. This suggests that the network topology might affect the performance of epidemic data dissemination protocols in a more complex network structure. In order to try to answer this question, we analyze the performance of a more realistic network topology using the Barabasi and Waxman topologies [17] which are employed as a representative of the Internet network. We make use of the network topology generator BRITE [10] to create such topology structures. Using the same conventional timestamped anti-entropy epidemic algorithm, we evaluate its performance over these complex network topologies. The evaluations show that better performance of the epidemic algorithm is achieved in the Barabasi topology model rather than the Waxman model. These results strongly suggest a topology dependance performance mainly due to the existing difference in the degree of connectivity. Exploring the impact of these network topology properties, we propose a new epidemic strategy for disseminating information in a more efficient fashion called topology-sensitive epidemic algorithm. This novel data dissemination strategy will prove to perform much faster than the conventional timestamped anti-entropy epidemic (TSAE) algorithm.

The paper is organized as follows. Section 2 describes the system model, specifying the assumptions we make and defining our framework. In section 3, we

outline the existing timestamped anti-entropy epidemic algorithm approach for data dissemination of information and we introduce the proposed topology-sensitive epidemic algorithm. Simulation results showing the performance of the TSAE and the proposed epidemic algorithms over different topology structures are presented in Section 4. Finally, some conclusions are drawn in Section 5.

2 System Model

This section describes the model that we rely on throughout this paper. We consider a model of a distributed network system consisting of N elements (nodes). Each of the N elements of the network can communicate only by exchanging a message as it is shown in Fig. 1. For simplicity, the discussion in this paper focuses on an asynchronous fully replicated system where reliable communication is assured, *i. e.* a message sent by node n_i to node n_j for $i, j = 1, 2, \cdots, N$ is eventually received by node n_j. Every node is considered to be a server site which provides a service to a number of local clients, see Fig. 1. The clients can make requests to a server where every request may be a read or write operation, or both. When a server receives a write operation request from a client, this operation is also propagated to all other servers (replicas) to guarantee the consistency of the distributed system. Moreover, if a client changes any object in a server by requesting a write operation then an update process, a message sent to the other replicas, needs to be carried out to all other servers in the network system.

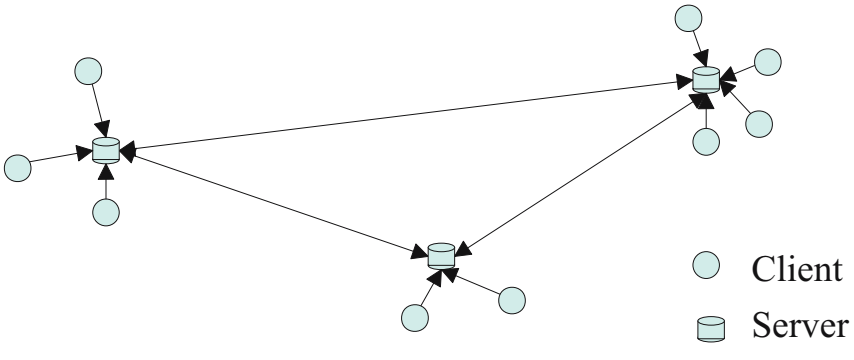


Fig. 1. Client-server network model under consideration

3 Epidemic Algorithms

Epidemic algorithms are used in a wide set of applications, *i.e* maintaining the consistency of distributed database systems (see [11] and [13]), routing in Mobile Ad Hoc networks [5], collaborative peer-to-peer applications [14], etc. Epidemic mechanisms follow the model of nature to spread information and define simple rules for information to flow between nodes of a network. Different variants of

epidemic dissemination algorithms exist, each with its own distinctive characteristics to disseminate information among a large number of processes with a dynamic connection topology. Among those main characteristics are:

- *Self-organization*: nodes only need to know about a few neighbors nodes. Generally, epidemic algorithms require few protocols sessions (short time) to update a distant node through the data dissemination mechanism without neither a central system controller nor topological system structure knowledge.
- *Fault-tolerance*: all active nodes will eventually receives the updated messages from others nodes despite of failed nodes and/or links in the network.
- *Scalability*: every node only needs to know a few neighboring nodes regardless the total number of nodes in the system.
- *Autonomy*: A server node can still provide or maintain the services to the group of clients connected to it even if there is not a temporal link between this server node and the rest of the network.

In this work, we select the already proposed timestamped anti-entropy epidemic algorithm used for the distributed bibliography database system presented in [9]. In what follows, we recall this epidemic protocol. Following this, we proposed two different epidemic algorithms that exploit the topology properties of a distributed system. The detailed description of such algorithms are presented next.

3.1 Timestamped Anti-entropy Epidemic Algorithm

In this section, we briefly describe the timestamp based anti-entropy epidemic (TSAE) algorithm. A timestamp represents the temporal information related to any change requested by a client in the objects kept for a particular node. Other important data structure is the message log vector representing the temporal buffer where the new messages in a node are stored.

The first step for the timestamped anti-entropy protocol is selecting which neighbor to exchange information with. In this case, this selection of the neighbor is done randomly from the collection of neighbors in the range of the node. Then the messages initiated from the source are re-sent by neighboring nodes, extending outward node by node until the entire network nodes have received the messages. This process is accomplished by exchanging periodically messages between a pair of nodes. The data exchange between nodes follows a predefined structure which is shown in Fig. 2 and 3. This process is described below:

- A session between two nodes starts when a session request is accepted from any of the nodes in communication.
- The next step is to exchange their summary timestamp and message acknowledgment vectors. Timestamps, which represent a snapshot of the communication state in the system, are used to provide an ordering upon events within the system [16]. The messages received at each node are kept in a message log.

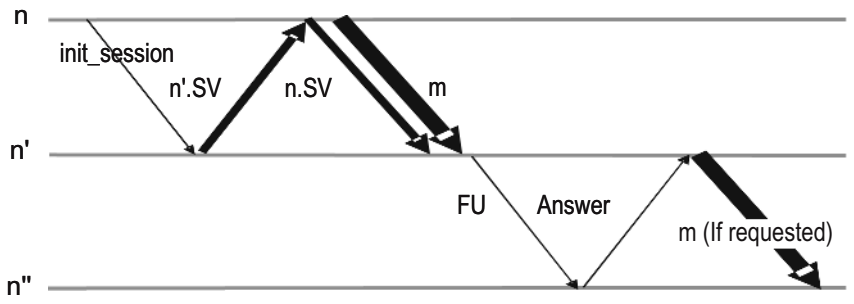


Fig. 2. Exchanging information process by the timestamped anti-entropy epidemic protocol and proposed topology-sensitive epidemic algorithm. Where n, n'and n" are neighbor nodes, SV is the summary vector, FU is a query for Fast Update process and **m** represents a message.

- Each node maintains a complete copy of all objects in the replication system by checking its summary timestamp.
- When a failure situation occurs during a message exchange process, any node can abort the session process and the state of the nodes can be discarded. A session process ends in a message acknowledgment exchange. In a successful session, both nodes have received the same set of messages.

3.2 Topology-Sensitive Epidemic Algorithms

The principle that characterize the epidemic data dissemination protocol described in the previous section, is mainly concerned with the random selection of the neighbor node. No special care has been put into the selection of nodes according to some criteria, for example depending on the network topology structure. Epidemic algorithms can be differentiated from each other by their style of communication between neighboring nodes. In this paper, we propose then to modify the timestamped anti-entropy epidemic algorithm by introducing a node selection criterion which is based on the network topology. Unlike random node selection, the proposed epidemic algorithm chooses the neighbor node located at distance 1 from the source, distance 1 refers to those nodes that requires only one hop to reach the source node, with greater out-degree. We consider the term out-degree of a node as the number of links connected to it. For this algorithm, it is assumed that each node knows the group of neighbors at distance 1 and their corresponding number of links of each node (out-degree). Notice that it is not necessary for the algorithm to know the entire topology of the network. This characteristic yields a high degree of availability, autonomy and scalability of the proposed epidemic algorithm for data dissemination in any large-scale networks.

Taking advantage of the network topology, another characteristic we exploit in the proposed topology-sensitive Epidemic Algorithms is through a fast update (FU) process as shown in Fig. 2 and 4. In this FU process, when a server node receives a new message, it immediately sends a new message to any neighbor node

```
after n.wait(random time)
    n'=n.select_neighbor(n.neigbors[ ],in random way)
    n.send_init_session(n') // message sent to n'
    when n'.receive_init_session:
        n'.send_SV(n,n'.SV[ ]) //SV: summary vector
    when n.receive_SV:
        n.send(n',n.SV[ ])
        //Calculate diffV==messages n' not yet received
        diffV=n.compareSV(n.SV[ ],n'.SV[ ])
        foreach message m in diffV[ ]:
              n.send(n',m)//send message m to n'
              when n'.receive(n.SV[ ]):
              diffV[ ]=n.compareSV(n'.SV[ ], n.SV[ ])
        foreach message m in diffV[ ]:
              n'.send(n, m)
    when n'.receive(m):
              n'.process(m) //and update n'.SV[ ]
    when n.receive(m):
              n.process(m) //and update n.SV[ ]
```

Fig. 3. Pseudo-code for the basic epidemic algorithm

at distance 1 and with out-degree greater that the node itself. If the neighbor node has already the message, the session is ended immediately, otherwise the message is sent to this neighbor node. Then this neighbor node repeats the same process to propagate the new message to its own neighbor nodes at distance 1 and with greater out-degree. This process is repeated until all reachable network nodes with high degree of connectivity has received the new messages.

4 Simulation Performance

4.1 TSAE Performance in Simple Topology Networks

In order to observe the behavior of the epidemic protocol TSAE, some simulations have been carried out to show the topology network influence over the epidemic algorithms. Firstly, a TSAE algorithm simulator has been built using Network Simulator 2 [15]. Three different types of simple topologies are evaluated: 1) ring 2) line and 3) mesh. The experiments are carried out with 5000 trials with a confidence index equal to 99% for each topology.

It is assumed that all nodes contain a new message by the time the experiment is set to start, *i.e.* the system is in a non-consistent state. In order to reach a consistent state in the network system, all nodes must collect all new messages generated by the other nodes in the rest of the network. It is also assumed that the network system is in a maximum stress, each node in the network holds a non-consistent state status, at the beginning of the simulations.

```
after n.wait(random time)
     n'=n.select_neighbor(n.neigbors[ ],HIGHEST_OUTDEGREE)
     n.send_init_session(n')// message sent to n'
     when n'.receive_init_session:
         n'.send_SV(n,n'.SV[ ]) //SV: summary vector
    when n.receive_SV:
         n.send(n',n.SV[ ])
        //Calculate diffV==messages n' not yet received
        diffV[ ]=n.compareSV(n.SV[ ], n'.SV[ ])
        foreach message m in diffV[ ]:
               n.send(n',m)//send message m to n'
     when n'.receive(n.SV[ ]):
         diffV[ ]=n.compareSV(n'.SV[ ], n.SV[ ])
         foreach message m in diffV[ ]:
               n'.send(n, m)
         when n'.receive(m):
               n'.process(m) //and update n'.SV[ ]
         foreach neighbor e in n'.neighbors[ ]
               where n'.neighbors[e].out-degree > n'.out-degree:
               FU=new fast_update(m.id, m.tstmp)
               n'.send_fast_update(e,FU)
         when n".receive_fast_update(FU):
               if n".check_history(FU) == UNKNOWN):
                       n".request(FU.id)
               else n".reject(FU.id)
         when n'.receive(FU.id): //if requested
               n'.send(n", m) //send message m to n"
```

Fig. 4. Pseudo-code for the proposed topology-sensitive epidemic algorithm

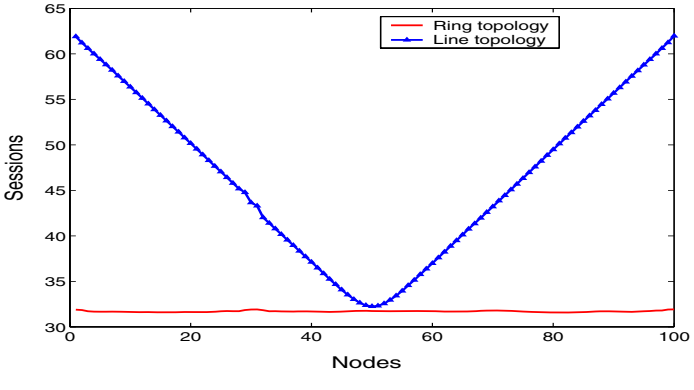


Fig. 5. TSAE performance using a line and ring topologies

Figure 5 illustrates the dynamics behavior of the TSAE algorithm through the ring and line topologies. These simulation results show the number of sessions required for each node to receive all new messages from the network. We observe that in the ring topology case, each node reach a consistent state in approximately the same number of sessions. In contrast, the time to reach a consistent state for every node in the line topology depends on its location, the farther away from the center of the network the higher of the number of sessions that are required. Therefore, it is clear then that the consistent state of the network is reached faster through a ring topology. In the same way, Fig. 6 presents the results using now a mesh topology. A mesh of 17×17 nodes is considered. From this Figure, we can see the same performance trend to those topologies analyzed in Fig. 5.

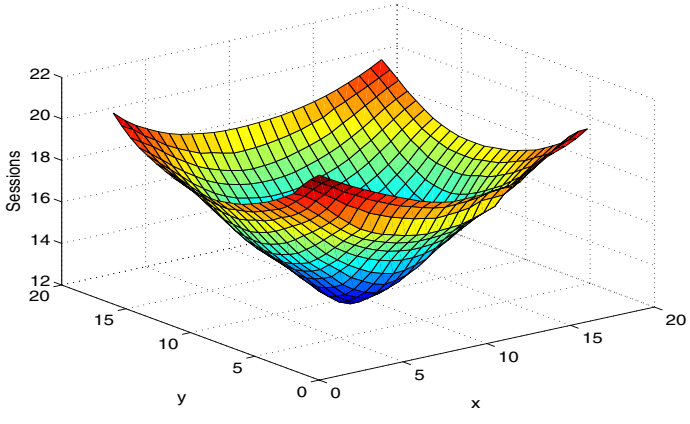


Fig. 6. TSAE performance using a mesh topology, 17×17 nodes

4.2 TSAE Performance in Complex Topology Networks

In the previous section, we have studied the TSAE performance over simple network topologies. This assumption is, however, not valid anymore in the case of a more complex scale-free networks, where it is known that spreading processes may show very different properties. To explore furthermore the topology network influence over the TSAE algorithm, we evaluate the performance of this process employing the Barabasi [17] and Waxman topology models which are used as topologies representative of the the Internet network. Waxman is one of the first developed network topology generators, it is a geographic model for the growth of a computer network which distributes nodes randomly over a rectangular coordinate grid. In this model the nodes of the network are uniformly distributed and edges are added according to probabilities that depend on the distances between the nodes. The probability to have an edge between nodes u and v is given by

$$P(u, v) = a \exp\left(-\frac{d}{bL}\right)$$

where $0 < a, b \leq 1$, d is the distance from u to v, and L is the maximum distance between any two nodes. Observe that the probability of edges between any two nodes increases with the parameter a. On the other hand, one of the most interesting features of a large class of the complex networks is their scale-free behavior: each node of the network is connected to some other k nodes. The number of connections obeys a power-law distribution, i.e. $P(k) \sim k^{\gamma}, 2 \leq \gamma \leq 3$ for most networks considered. Such networks are called scale-free because the fluctuations of the distribution around the average value k are infinite (they do not possess any particular scale). The difference between a scale-free network and a random network (where every link between different nodes is present with a probability p) hints towards some mechanisms that generated the observed network features. One of the most celebrated models that explains the emergence of scale-free networks is the Barabasi model. According to the Barabasi model, the two essential ingredients for the formation of scale-free networks are growth and preferential attachment. Growth implies that new nodes are added to the network over time at a more or less constant rate. Preferential attachment means that a newly added node connects preferentially to nodes that already have a high degree: a new node tries to attach to authoritative nodes and the degree of a node is an effective representation of its authoritativeness.

We have generated Waxman and Barabasi topologies using BRITE [10], a network topology generator, then were feed to our simulator, written on Network simulator [15]. The simulations were run for 5000 experiment trails with a confidence index equal to 0.99. As before, the systems were set to be in a non-consistent state at the beginning of the experiment.

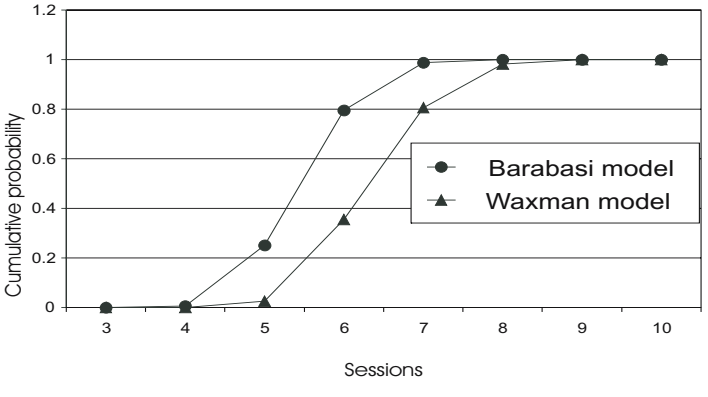


Fig. 7. TSAE performance using a Barabasi and Waxman complex topology model

Figure 7 demonstrate the performance difference of the TSAE algorithm as a function of network topology. This is illustrated by plotting the cumulative probability function as a function of the number of sessions. We can observe that the probability of reaching a consistent state by the TSAE algorithm in 7 sessions is almost of 1 with the Barabasi model while a probability of 0.8

is obtained for the Waxman model. In general, the TSAE algorithm performs better (faster propagation process) over the Barabasi topology model than the Waxman model. It indicates that the topology structure strongly affects the propagation mechanism of the TSAE algorithm. In this case, the performance difference is yielded by the degree of connectivity in both topology structures.

4.3 Proposed Epidemic Algorithm Performance Using the Barabasi Model

In this section, we evaluate the performance of the proposed epidemic algorithm using the Barbarasi model since it provides a topology network representative of Internet (scale-free network). Using the same assumptions and simulation parameters as presented in the previous section, we show in Fig. 8 the results for the proposed topology-sensitive epidemic algorithm. As before, the epidemic algorithm performance is measured in terms of the cumulative probability for the number of sessions required to reach the consistent state in the network system. It is observed in Fig. 8 that there is a probability of almost 1 for the proposed epidemic algorithm to reach a consistent state in the network in less than 5 sessions. In contrast, a consistent state in the network is reached with a probability of 1 by the TSAE algorithm in approximately 9 sessions. It is then clear that a faster information spreading process can be achieved by the proposed epidemic algorithm. We believe this improvement is obtained mainly by a number of high out-degree nodes which increase the network connectivity. Since the number of connections obeys a power-law distribution, the number of nodes with high out-degree results to be smaller than those with low out-degree. Although there is a smaller number of high out-degree nodes, this property helps to improve the connectivity of the entire network. Exploiting such characteristics, the proposed algorithm is designed to send the data in a preferential fashion to the nodes with high out-degree allowing a faster information spreading process.

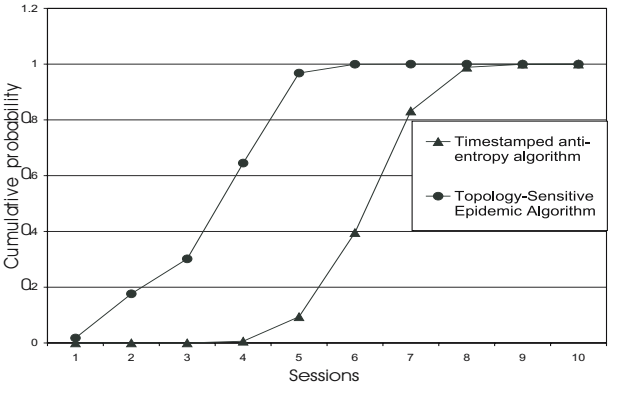


Fig. 8. Performance comparison of the proposed topology-sensitive epidemic algorithm versus TSAE algorithm over a scale-free topology

5 Conclusions

In this paper, we have studied the effect of network topology in epidemic algorithms for the information spreading process. Through simulations, we have firstly presented the behavior and properties of the TSAE epidemic protocol over simple network topologies, results for a line, ring and mesh topologies were presented. We showed that, for the purpose of information dissemination, the TSAE performance was clearly affected by the topological structure of the network. Simulations of the TSAE algorithm over a more complex topology structure showed similar results. The obtained results indicate the seeking of epidemic algorithm mechanisms which take into account the specific topology of the underlying network. In order to provide with a more efficient data dissemination process, we introduced a new topology-sensitive epidemic algorithm aimed at large-scale networks. This approach exploits a local knowledge of the network topology in each server node so that data can be propagated faster. A comparison of the proposed topology-sensitive algorithm with the TSAE algorithm was carried out in an Internet-like topology structure. Performance simulations showed that the proposed epidemic algorithm obtained a significant improvement as compared to the TSAE epidemic algorithm. These results stimulate the applicability of epidemic algorithms in technological and communication networks.

Acknowledgement

This research has been partially supported by the Mexican Ministry of Education under contract P/PROMEP/103.5/03/2557 and the FAI-UASLP, México.

References

1. R. Pastor-Satorras and A. Vespignant. *Epidemic Spreading in Scale Free Networks*, Physica Review Letters, 86, 2001.
2. I. Pool and M. Kochen. *Contacts and influence*. Social Networks, 1:5(51, 1978).
3. F. Wu, B. A. Huberman, L. A. Adamic, and J. R. Tyler. *Information Flow in Social Groups*. Annual CNLS conference on Networks, Santa Fe, NM, May 12, 2003. http://www.hpl.hp.com/shl/papers/flow/.
4. P. T., R. G., A. K., L. M. *Epidemic Information Dissemination in Distributed Systems*. Computer, May 2004, page(s): 60- 67. Volume: 37, Issue: 5.
5. L. Li, J. halpern, Z. J. Hass: Gossip-based ad hoc routing, Proceedings of Infocom, 2002, pp. 1707-1716.
6. F. M. Cuenca-Acuna, C. Peery, R. P. Martin, T. D. Nguyen. *Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities*. In Procidings of the IEEE International Symposium on High Performance Distributed Computing (HPCD 12), Seattle, WA, June 2003.
7. A. Ganesh and A. Kermarrec and L. Massoulie. *Peer-to-Peer Membership Management for Gossip-based Protocols*. IEEE Trans. Comp., 52(2):139–149, February 2003.
8. I. Foster, C. Kesselman, S. Tuecke. *The Anatomy of the Grid: Enabling Scalable Virtual Organizations International J. Supercomputer Applications*. 15(3), 2001.

9. R. A. Golding. *Weak-Consistency Group Communication and Membership*. PhD thesis, University of California, Santa Cruz, Computer and Information Sciences Technical Report UCSC-CRL-92-52, December 1992.
10. *BRITE*. The Boston Representative Internet Topology Generator: http://www.cs.bu.edu/brite/.
11. A. Adya. *Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions*. PhD thesis M. I. T., Department of Electrical Engineering and Computer Science. March 1999.
12. R. Guy, J. Heidemann, W. Mak, T. Page, Jr., G. Popek and D. Rothmeier. *Implementation of the Ficus Replicated File System*. Proceedings Summer USENIX Conf. June 1990.
13. JoAnne Holliday, Divyakant Agrawal, Amr El Abbadi. *Partial Database Replication using Epidemic Communication*. Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS02), Vienna 2002.
14. Joan Manuel Marques and Leandro Navarro. *Autonomous and Self-sufficient Groups: Ad Hoc Collaborative Environments*. 11th International Workshop on Groupware (CRIWG-2005) September, 2005
15. The Network Simulator 2: http://www.isi.edu/nsnam/ns/
16. L. Lamport. *Time, clocks, and the ordering of events in a distributed system*. Communications of the ACM, 21(7):558(1978).
17. Barabasi. A.-L. Barabsi and R. Albert. *Emergence of Scaling in Random Networks*. Science, pp. 509 512, Oct. 1999.
18. Faloutsos, G. Siganos, M. Faloutsos, P. Faloutsos, Ch. Faloutsos. *Power laws and the AS-level internet topology*. IEEE/ACM, Transactions on Networking, Volume 11, Issue 4 August 2003, pp. 514 524.
19. J. Acosta-Elas, U. Pineda, J.M. Luna-Rivera, E. Stevens, I. Campos-Canton, and L. Navarro-Moldes. *The Effects of the Network Topology over Epidemic Algorithms*. International Conference on Computational Science and Its Applications (ICCSA 2004), Assisi, Italy, May 14-17, 2004.

Performance Modeling and Optimal Block Size Selection for the Small-Bulge Multishift QR Algorithm

Yusaku Yamamoto

Nagoya University, Nagoya, Aichi, 464-8603, Japan
yamamoto@na.cse.nagoya-u.ac.jp
http://www.na.cse.nagoya-u.ac.jp/~yamamoto

Abstract. The small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias is one of the most efficient algorithms for computing the eigenvalues of nonsymmetric matrices on processors with hierarchical memory. However, to fully extract its potential performance, it is crucial to choose the block size m properly according to the target architecture and the matrix size n. In this paper, we construct a performance model for this algorithm. The model has a hierarchical structure that reflects the structure of the original algorithm and given n, m and the performance data of the basic components of the algorithm, such as the level-3 BLAS routines and the double implicit shift QR routine, predicts the total execution time. Experiments on SMP machines with PowerPC G5 and Opteron processors show that the variation of the execution time as a function of m predicted by the model agrees well with the measurements. Thus our model can be used to automatically select the optimal value of m for a given matrix size on a given architecture.

1 Introduction

The QR algorithm [1][2][3] is widely used as an efficient and reliable method to compute the eigenvalues of small to medium nonsymmetric matrices. However, it is not straightforward to implement the QR algorithm in a way that fully exploits the performance of modern computers such as the shared-memory parallel machines and processors with hierarchical memory. In fact, when applying the conventional double implicit shift QR algorithm to an $n \times n$ Hessenberg matrix, the parallel granularity is only $O(n)$. This is often too small to attain reasonable speedup in the presence of large inter-processor synchronization costs. In addition, the algorithm sweeps the whole matrix in each QR iteration, while performing only $O(1)$ arithmetic operations on each matrix element. This results in poor data reference locality and prevents effective use of cache memory.

Many efforts have been made to overcome these difficulties so far [4][5][6][7][8][9]. Among them, the small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias [5] seems the most promising approach. Their algorithm computes m shifts at once as the eigenvalues of the $m \times m$ trailing principal submatrix,

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 451–463, 2006.
© Springer-Verlag Berlin Heidelberg 2006

as in the conventional multishift QR algorithm [4], and performs so-called bulge chasing [10][11] using these m shifts simultaneously. However, instead of chasing one large bulge containing the m shifts, their algorithm chases $m/2$ small bulges each containing only 2 shifts simultaneously in a pipelined fashion. This removes the defects of conventional multishift QR algorithm — numerical instability due to the use of large bulge and the resulting deterioration of convergence speed [6][12] — and recovers the excellent convergence property of the original double shift QR algorithm. Since the parallel granularity and the number of operations per iteration and per matrix element are increased to $O(m^2n)$ and $O(m)$, respectively, as a result of using m shifts, this algorithm is ideally suited for modern architectures. It has been reported that this algorithm can achieve up to three times speedup over DHSEQR, the large-bulge multishift QR algorithm in LAPACK [13], on the Origin2000 [5].

To fully extract the potential performance of the small-bulge multishift QR algorithm, it is crucial to choose the block size m properly. Larger m will provide larger parallel granularity and more chance for data reuse, but the cost of computing the shifts grows with m. The optimal value of m varies depending on the target machine, the number of processors and the matrix size n. In principle, the optimal value of m can be determined by trial and error, but this process will require huge time.

In this paper, we present a performance model for the small-bulge multishift QR algorithm. It is based on the hierarchical approach proposed by Dacklund [14] and Cuenca et al. [15][16], and given the execution time models of each component of the algorithm, such as the level-3 BLAS routines and the conventional double shift QR algorithm to compute the shifts, predicts the total execution time accurately. Using this performance model, the optimal block size for a given architecture, the number of processors and matrix size can be determined prior to execution.

There are many studies on automatic optimization of linear algebra programs. Katagiri et al. [17] propose I-LIB, an automatically tuned linear algebra library for distributed-memory parallel machines. They report the result of optimizing parameters for tridiagonalization of symmetric matrices and show that considerable performance gains can be obtained through optimization. However, to find the optimal set of parameters, I-LIB measures the execution time of the whole program repeatedly with different values of parameters. Hence the cost of tuning is very high.

The hierarchical approach to performance modeling was first proposed by Dackland [14] and Cuenca [15][16]. The basic idea of this approach is to exploit the natural hierarchy existing in linear algebra programs. In this approach, the execution time models of lower-level routines such as the BLAS are constructed first, and the total execution time is estimated by accumulating the execution times of the lower-level routines. This approach has been applied to performance modeling and optimization of LU decomposition [15], QR decomposition [14], tridiagonalization of symmetric matrices [18] and so on and has proved useful in

reducing the cost of tuning without sacrificing accuracy. Our work is along this line of research.

Automatic optimization of linear algebra programs are studied both at a higher and a lower level. For example, the Self-Adapting Numerical Software (SANS) proposed by Dongarra et al. [19] aims at automatically selecting the best routine (not parameters within a routine) to solve a given problem. On the other hand, there are projects like ATLAS [20] and PhiPAC [21], which try to maximize the performance of basic routines such as BLAS by automatic tuning. These studies deal with problems that are at different levels from ours and are therefore complementary with our work.

This paper is structured as follows: in section 2, we give a brief explanation of the small-bulge multishift QR algorithm. Section 3 gives the details of our performance model. Experimental results that demonstrate the effectiveness of our model are presented in section 4. Finally, section 5 gives some concluding remarks.

2 The Small-Bulge Multishift QR Algorithm

2.1 The Algorithm

Let us consider the application of the QR algorithm on an $n \times n$ Hessenberg matrix A and denote the matrix after the l-th QR iteration by A_l. In the conventional double implicit shift QR algorithm, to compute A_{l+2} from A_l, we first compute the two shifts σ_1 and σ_2 as the eigenvalues of the trailing principal submatrix of A_l, introduce a 4×4 bulge containing the information of the shifts at the top left corner of A_l and then chase the bulge along the diagonal by repeatedly applying Householder transformations until it disappears from the bottom right corner. Then the implicit Q theorem [10][11] guarantees that the resulting matrix is the one that would be obtained by applying two explicit QR steps with shifts σ_1 and σ_2 to A_l.

This idea can be naturally extended to the multishift QR algorithm [4]. In this case, we compute m shifts $\sigma_1, \sigma_2, \ldots, \sigma_m$ as the eigenvalues of the trailing principal submatrix of A_l, introduce a large $(m+2) \times (m+2)$ bulge containing their information and then perform bulge chasing. By using a large bulge, both parallel granularity and data reference locality of the bulge-chasing step can be increased. The LAPACK routine DHSEQR is based on this idea. Unfortunately, it has been shown that as the size of the bulge increases, the shift information contained in the bulge becomes extremely prone to being contaminated by numerical errors [12]. This prevents the accurate shift information to be conveyed to the bottom right corner of the matrix and thereby retards convergence [12]. Thus the value of m is usually limited to about ten, but this is too few to use cache memory efficiently.

To overcome this difficulty, Braman, Byers and Mathias propose the small-bulge mltishift QR algorithm [5]. In this algorithm, the m shifts are divided into $m/2$ sets of double shifts and in the bulge-chasing process, $m/2$ small 4×4 bulges are chased simultaneously in a pipelined fashion. Although this algorithm

is mathematically equivalent to the large-bulge multishift QR algorithm, it can avoid numerical difficulties associated with the use of large bulge and recover the excellent convergence properties of the conventional double shift QR algorithm.

The bulge chasing in the small-bulge multishift QR algorithm can be divided into three phases. In phase I, a chain of $m/2$ bulges is introduced at the top left corner of the matrix. Since bulges have to be at least three rows apart to avoid interference, the bulges occupy rows 1 through $3m/2+1$ when the phase I is completed (Fig. 1). Note that to chase the bulges in phase I, only the first $(3m/2+1)\times(3m/2+1)$ submatrix of A_l, which we denote by $A_{l,1:3m/2+1,1:3m/2+1}$ following [11], is necessary. We therefore divide the work in phase I into two steps (see Fig. 2):

(a) Chase the bulges along the diagonal by applying a sequence of Householder reflections from both sides to $A_{l,1:3m/2+1,1:3m/2+1}$. At the same time, form the product of these reflections as an $(3m/2+1)\times(3m/2+1)$ matrix U.
(b) Update the rest of rows 1 through $3m/2+1$ by multiplying $A_{l,1:3m/2+1,3m/2+2,n}$ by U from the left.

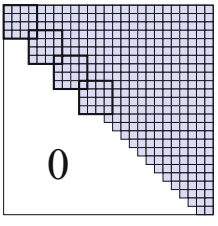


Fig. 1. The matrix after the completion of phase I ($m = 8$). Four bulges have been introduced in rows 1 through (3/2)*8+1.

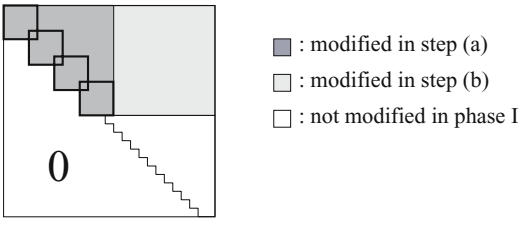


Fig. 2. Division of the work in phase I into two parts: (a) bulge-chasing in the diagonal block and (b) update of the off-diagonal block.

Of these two steps, step (b) can be done entirely with the level-3 BLAS, or matrix multiplication, and the copy operation needed to move the results back into $A_{l,1:3m/2+1,3m/2+2,n}$. Though step (a) cannot be performed with the level-3 BLAS, the computational work needed for this step is much smaller than that for step (b) when $m \ll n$. Thus most of the work in phase I can be organized as level-3 BLAS.

In phase III, the $m/2$ bulges that lie on the last $3m/2+1$ rows of A_l are chased out of the matrix. As in phase I, the work can be divided into two steps, namely, (a) bulge-chasing within the trailing $(3m/2+1) \times (3m/2+1)$ diagonal block and accumulation of the Householder reflectors, and (b) update of the off-diagonal block of the last $3m/2+1$ columns. Again, step (b) accounts for most of the computational work and can be done with the level-3 BLAS.

In phase II, the $m/2$ bulges that lie on the first $3m/2+1$ rows of the matrix are chased to the last $3m/2+1$ rows along the diagonal. In this phase, we set some integer k and regard the operation of chasing all the bulges by k rows as one block operation (Fig. 3). Since the sequence of the bulges occupies $3m/2+1$ rows and columns, this block operation involves $3m/2+k$ rows and columns. Then we divide the work in this block operation into three steps, that is, (a) bulge-chasing within the $(3m/2+k) \times (3m/2+k)$ diagonal block and accumulation of the reflectors, (b) update of the off-diagonal block of the $3m/2+k$ rows and (c) update of the off-diagonal block of the $3m/2+k$ columns. Again, steps (b) and (c) account for most of the work and they can be done with the level-3 BLAS. Braman et al. [5] shows that $k \sim \frac{3}{2}m$ is the best to minimize the computational work of phase II.

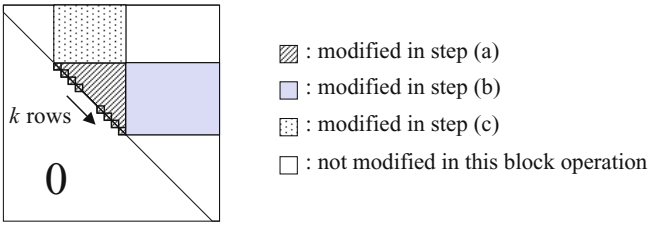


Fig. 3. Work in one block operation of phase II. The work is divided into (a) bulge-chasing in the diagonal block, (b) update of the off-diagonal rows and (c) update of the off-diagonal columns.

In addition to the work in phases I to III described above, there is work for computing the shifts. Also, when the matrix A_l becomes sufficiently small as a result of deflation, its eigenvalues are computed with the conventional double implicit shift QR algorithm. However, the computational work associated with them are much smaller than the work in phases I to III when $m \ll n$. Thus the small-bulge multishift QR algorithm can perform most of the work in the form of level-3 BLAS and exploit the potential performance of modern architectures. Numerical experiments on various test matrices show that it can attain up to three times speedup over DHSEQR on the Origin2000 [5].

2.2 Basic Computational Routines Used in the Algorithm

As is clear from the previous subsection, the small-bulge multishift QR algorithm consists of four types of basic computational routines as follows:

(A) Routines for chasing the bulges within the diagonal block. We denote the routines for phases I, II and III by BCHASE1, BCHASE2 and BCHASE3, respectively.
(B) level-3 BLAS routines for updating the off-diagonal block of the rows or columns. We can use DGEMM with TRANSA=TRANSB='N' for row updates and DGEMM with TRANSA='N' and TRANSB='T' for column updates.
(C) Two copy routines which we denote as COPY1 and COPY2. The former is used to copy the result of row updates back into the matrix, while the latter is used to copy the result of column updates back into the matrix.
(D) A Double implicit shift QR routine for computing the shifts or computing the eiegnvalues of A_l when A_l is sufficiently small. We use EISPACK routine HQR for this purpose.

In the hierarchical performance modeling to be explained in the next section, we use these routines as basic components.

2.3 The Optimal Block Size

The attain high performance with the small-bulge multishift QR algorithm, it is critical to choose the optimal block size m. In general, as m becomes larger, the performance of the level-3 BLAS will increase because both the parallel granularity and the chance for data reuse increase. On the other hand, the cost of computing the shifts and the work of BCHASE1 to 3 relative to the total work will also grow with m. The optimal value of m is determined from these trade-offs and differs considerably depending on the architecture, the number of processors and the matrix size n. In fact, to obtain the best performance on the Origin2000, Braman et al. use $m = 60$ when $1000 \leq n \leq 1999$, $m = 116$ when $2000 \leq n \leq 2499$ and $m = 150$ when $2500 \leq n \leq 3999$ [5]. Our objective is to determine the optimal value of m for a given environment and problem size prior to execution using a performance prediction model.

3 Performance Modeling

3.1 The Hierarchical Approach

To construct a performance model of the small-bulge multishift QR algorithm, we adopt the hierarchical modeling approach [16][15][14]. In this approach, we first construct execution time models for the basic components of the algorithm such as the level-3 BLAS routines and the double implicit shift QR routine based on the measurement of actual execution times. Then, the time consumed by each call to these subroutines in the algorithm is estimated using the model and the input parameters. Finally, the execution time of the whole algorithm is predicted by accumulating these partial execution times. This methodology has been applied to the performance modeling of LU and QR decompositions and a BLAS-3 based tridiagonalizaion algorithm and has proved to give satisfactory results both in terms of accuracy and cost of prediction.

3.2 Modeling the Performance of Basic Computational Routines

From what we have stated in subsection 2.2, we need to construct execution time models for eight computational routines, namely, BCHASE1, BCHASE2, BCHASE3, DGEMM('N','N'), DGEMM('N','T'), COPY1, COPY2 and HQR. Here we take up the case of BCHASE2 to illustrate the process of modeling. This routine is used to chase the bulges within the diagonal block in phase II and has two input parameters m and k that determine the execution time. Our aim is to model the execution time of this routine as a function $f_{\mathrm{BCHASE2}}(m,k)$ of m and k. Note that the two-parameter model is necessary even when k is fixed. This is because the number of rows by which the bulges are chased in phase II is in general not a multiple of k and the remainder part must be processed by BCHASE2 with a smaller value of k.

To construct a model, we first measure the performance of BCHASE2 on grid points in the (m,k) plane. More specifically, we vary m from 10 to 150 with intervals of 10 and set k to one of the five values, $\frac{1}{5}m$, $\frac{2}{5}m$, $\frac{3}{5}m$, $\frac{4}{5}m$ and m. Then we approximate the execution time for a fixed value of k as a cubic function of m:

$$f_{\mathrm{BCHASE2}}(m,k) = f^{(k)}_{\mathrm{BCHASE2}}(m) = a_3^{(k)}m^3 + a_2^{(k)}m^2 + a_1^{(k)}m + a_0^{(k)}. \quad (1)$$

The coefficients $a_3^{(k)}, a_2^{(k)}, a_1^{(k)}$ and $a_0^{(k)}$ are determined by the least squares from the measured data. To obtain the value of $f_{\mathrm{BCHASE2}}(m,k)$ for k between the grid points, we compute the function values at the two adjacent grid points and use linear interpolation. It would be possible to use polynomial approximation also with respect to k, but we chose to use linear interpolation because it is more flexible and can approximate the function well even when it exhibits somewhat irregular behavior due to cache miss.

The performance modeling of other routines can be done in much the same way. In fact, COPY1 and COPY2 also have two parameters that affect the execution time. DGEMM('N','N') and DGEMM('N','T') also have only two parameters since in our algorithm, the multiplier U is a square matrix. BCHASE1, BCHASE3 and HQR have only one parameter and therefore their performance modeling is easier.

3.3 Modeling the Performance of the Whole Algorithm

Once execution time models of the basic computational routines have been constructed, we can use them to predict the execution time of the small-bulge multishift QR algorithm. The conventional approach to this is to derive an analytical expression for the total computational work executed by each routine and then calculate the time consumed by each routine using the corresponding performance models. However, it is difficult to obtain accurate results with this approach because the relation between the execution time and the computational work is usually far from linear.

We therefore take an alternative approach. We first write functions named BCHASE1_TIME, DGEMM_TIME, COPY1_TIME, HQR_TIME and so on. These

functions have the same input parameters as BCHASE1, DGEMM, COPY1 and HQR, respectively, but instead of doing actual computation, estimate the execution time for given input parameters using the performance model for the corresponding routine and return it. Next we rewrite the main program of the small-bulge multishift QR algorithm so that the calls to the computational routines are replaced with the calls to the corresponding time estimation routines and the estimated execution times are accumulated. Then, by executing the rewritten program, we can obtain estimated execution time of the small-bulge multishift QR algorithm for a given value of n and m. This approach has been successfully applied to the performance modeling of a BLAS-3 based tridiagonalization algorithm. [18] reports that it can predict the execution time of this algorithm for various matrix sizes and block sizes on different architectures within errors of 5 to 10%.

Since the QR method is an iterative method, we have to address one problem that does not exist in the case of the tridiagonalization algorithm. We have to specify how many times the main loop is executed before convergence and in what manner the deflations occur. Kressner [22] observes that in average, deflation occurs after every four multishift sweeps and at each deflation, approximately $m \times m$ trailing submatrix is isolated. We confirmed this observation through our experiments on various matrices and adopt it as a model of convergence behavior in our performance estimation program.

Our performance model takes fully into account the nonlinearity between the execution time and computational work of the basic computational routines. Thus it is expected to predict the total execution time accurately, except for the variation due to variation in the number of iteration. The cost of prediction is proportional to the number of calls to the basic computational routines in the algorithm and is $O(n^2/m^2)$. This is negligible compared with the computational work needed for actual execution of the algorithm, which is $O(n^3)$. Note also that our performance model can be applied to shared-memory parallel programs without difficulty as long as parallelization is done within the basic computational routines.

4 Experimental Results

4.1 Computational Environments

To evaluate the effectiveness of our performance model, we performed experiments on two platforms, namely, a 2way SMP machine with 2.0GHz PowerPC G5 processors and a 4way SMP machine with 1.8GHz Opteron processors. For the G5 machine, we used IBM XL Fortran with options -O3 -qsmp=omp -qarch=ppc970 -qtune=ppc970 and the GOTO BLAS. For the Opteron machine, we used GNU f77 compiler with option -O4 and the GOTO BLAS. The routines BCHASE1, BCHASE2, BCHASE3, COPY1 and COPY2 were written from scratch and the EISPACK routine HQR is used for computing the shifts and for computing the eigenvalues of the matrix when it becomes sufficiently small. Parallelization is done only within the GOTO BLAS.

4.2 Performance Prediction

To construct the performance model, we first measured the execution times of the basic computational routines on both machines for various values of input parameters. For BCHASE1, BCHASE2, BCHASE3 and HQR, we varied m from 10 to 150 with intervals of 10. Performance measurement of BCHASE2 was done as described in subsection 3.2. For DGEMM('N','N') and DGEMM('N','T'), the size of the square multiplier matrix was varied from 10 to 300 with intervals of 10, while the number of columns in the multiplicand matrix was varied from 100 to 1000 with intervals of 100. The execution times of all the routines except for DGEMM were measured on 1 processor. The execution times of DGEMM were measured on 1 and 2 processors (in the case of PowerPC G5) or on 1 and 4 processors (in the case of Opteron). Based on these measurements, we built an execution time model for each routine on each platform following the prescription of subsection 3.2.

Next we constructed a performance model for the small-bulge multishift QR algorithm following the methodology stated in subsection 3.3 and compared the predicted execution time with the actual execution time. We varied the matrix size m from 1000 to 8000 and varied the number of shifts m from 30 to 120. The value of k was set equal to m. As test matrices, we generated random matrices whose elements follow the uniform distribution in $[-1, 1]$ and transformed them to Hessenberg by Householder transformation.

The results on the PowerPC G5 are shown in Table 1 and Fig. 4 for the 1-processor case and in Table 2 and Fig. 5 for the 2-processor case. It is clear that the model generally overestimate the actual execution time. This is because the test matrices used here require smaller number of iterations than we assumed in subsection 3.3. In fact, the average number of multishift QR sweeps needed to isolate an (approximately) $m \times m$ small submatrix was between 3 and 4. However, when we turn our attention to the relative execution time, defined as the ratio of the execution time to the shortest execution time over all m, we see that the model reproduces the behavior of the actual execution time fairly well (See Figs. 4 and 5). This is sufficient for determining the optimal value of m.

Table 1. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (PowerPC G5, 1CPU)

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	5.84	5.27	5.12	6.75
	8.28	6.79	6.77	7.82
2000	42.21	33.75	32.83	34.47
	53.40	40.53	39.59	46.09
4000	356.85	267.65	236.20	260.97
	393.86	288.60	267.22	296.86
8000	3073.24	2496.06	2270.95	2129.30
	3075.62	2180.10	1969.56	2116.30

Table 2. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (PowerPC G5, 2CPU)

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	4.51	3.94	4.51	6.02
	7.06	5.80	5.92	7.07
2000	33.33	24.74	24.99	29.25
	42.86	31.37	31.06	37.69
4000	274.19	190.05	176.53	207.83
	307.30	208.87	191.45	220.91
8000	2448.36	1812.72	1676.86	1807.11
	2370.45	1520.44	1328.71	1463.86

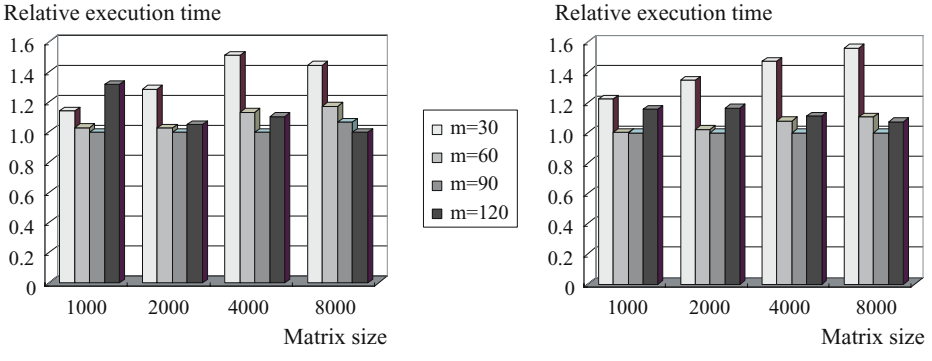


Fig. 4. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (PowerPC G5, 1CPU)

Table 3. Actual (above) and predicted (below) execution times (in sec.) of the small-bulge multishift QR algorithm (Opteron, 4CPU)

n	$m = 30$	$m = 60$	$m = 90$	$m = 120$
1000	3.58	3.80	4.37	5.56
	4.26	4.32	5.02	6.00
2000	24.32	20.86	24.67	24.18
	27.38	24.39	26.26	29.95
4000	189.38	144.14	141.81	145.77
	190.86	158.50	157.95	168.69
8000	1522.24	1069.79	1021.83	1014.12
	1419.33	1116.15	1066.48	1089.39

We also show the results on the 4way Opteron SMP machine in Table 3 and Fig. 6. In this case, again, the model generally overestimates the execution time, but predicts the relative execution time as a function of m fairly well.

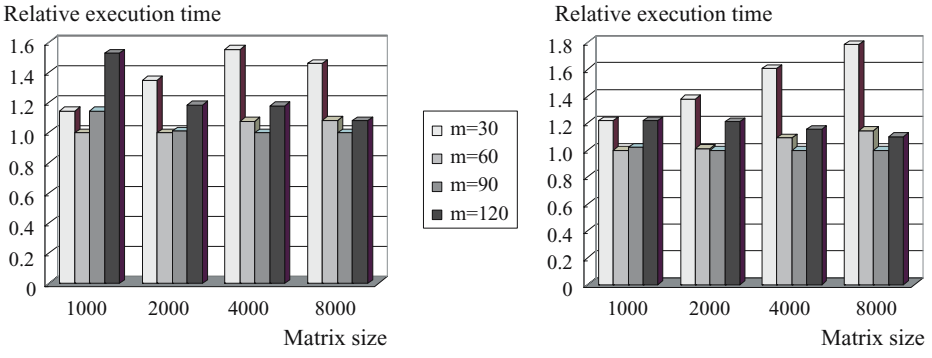


Fig. 5. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (PowerPC G5, 2CPU)

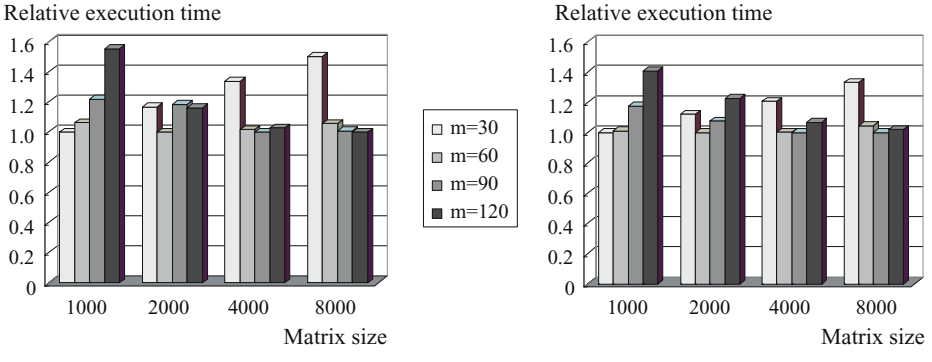


Fig. 6. Actual (left) and predicted (right) relative execution times of the small-bulge multishift QR algorithm (Opteron, 4CPU)

4.3 Optimal Block Size Selection

As can be seen clearly from Figs. 4 through 6, the value of m that gives the shortest execution time varies considerably depending on the matrix size and the architecture. Generally speaking, the optimal value of m increases with the matrix size. Also, the optimal value differs widely with the matrix size in the case of the 4way Opteron machine, while it is rather insensitive in the case of G5. Figs. 4 through 6 show that these tendencies are represented by our model well. In fact, our model succeeds in predicting the optimal value of m in 9 cases of the total 12 cases presented here. Even when it fails to predict the correct m, it can be seen that the execution time using predicted m differs only slightly from the shortest execution time. In addition, the time needed to predict the execution time for all value of m for given n is less than a fraction of a second. Thus we can conclude that our model can be used effectively for choosing the optimal block size in the small-bulge multishift QR algorithm.

5 Conclusion

In this paper, we construct a performance model for the small-bulge multishift QR algorithm proposed by Braman, Byers and Mathias. Our model has a hierarchical structure that naturally arises from the structure of the original algorithm and given the matrix size n, the number of simultaneous shifts m and the performance data of the basic components of the algorithm, such as the level-3 BLAS routines and the double implicit shift QR routine, predicts the total execution time. Experiments on SMP machines based on PowerPC G5 and Opteron processors show that the variation of the execution time as a function of m predicted by the model agrees well with the measurements. Thus our model can be used to automatically select the optimal value of m for a given matrix size on a given architecture.

Future works include extension of this model to a more efficient parallel implementation of the small-bulge multishift QR algorithm, where the bulge chasing in the diagonal block is overlapped with the update of the off-diagonal blocks. In this case, the number of parameters to optimize will increase and we will need an efficient search algorithm that can find near-optimal parameters without an exhaustive search of all possible candidates. As another direction of research, we are planning to extend the modeling methodology used in this work to distributed-memory parallel programs.

Acknowledgements

I would like to thank Professor Reiji Suda, Professor Takahiro Katagiri, Professor Toshitsugu Yuba, Professor Toshiyuki Imamura, Dr. Ken Naono and other members of the Auto-Tuning Study Group for fruitful discussion. This work is partially supported by the Ministry of Education, Science, Sports and Culture, Grant in Aid for Scientific Research on Priority Areas, "i-explosion" (No. 18049014), Grant-in-aid for Scientific Research (C)(No. 18560058) and Grant-in-Aid for the 21st Century COE "Frontiers of Computational Science".

References

1. Francis, J.G.F.: The QR transformation. a unitary analogue to the LR transformation. I. Comput. J. **4** (1961/1962) 265–271
2. Francis, J.G.F.: The QR transformation. II. Comput. J. **4** (1961/1962) 332–345
3. Kublanovskaya, V.N.: On some algorithms for the solution of the complete eigenvalue problem. U.S.S.R. Comput. Math. and Math. Phys. **3** (1961) 637–657
4. Bai, Z., Demmel, J.: On a block implementation of Hessenberg QR iteration. Int. J. of High Speed Computing **1** (1989) 97–112
5. Braman, K., Byers, R., Mathias, R.: The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance. SIAM Journal on Matrix Analysis and Applications **23** (2002) 929–947
6. Dubrulle, A.: The multishift QR algorithm: Is it worth the trouble? Palo Alto Scientific Center Report G320-3558x, IBM Corp. (1991)

7. Henry, G., Watkins, D.S., Dongarra, J.: A parallel implementation of the nonsymmetric QR algorithm for distributed memory architectures. SIAM J. Sci. Comput. **24** (2002) 284–311
8. Watkins, D.S.: Bidirectional chasing algorithms for the eigenvalue problem. SIAM J. Matrix Anal. Appl. **14** (1993) 166–179
9. Watkins, D.S.: Shifting strategies for the parallel QR algorithm. SIAM J. Sci. Comput. **15** (1994) 953–958
10. Demmel, J.W.: Applied Numerical Linear Algebra. SIAM (1997)
11. Golub, G.H., van Loan, C.F.: Matrix Computations. Third edn. Johns Hopkins University Press (1996)
12. Watkins, D.S.: The transmission of shifts and shift blurring in the QR algorithm. Linear Algebra and Its Applications **241/243** (1996) 877–896
13. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Croz, J.D., Greenbaum, A., Hammarling, S., McKenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide. SIAM (1992)
14. Dackland, K., Kågström, B.: A hierarchical approach for performance analysis of ScaLAPACK-based routines using the distributed linear algebra machine. In: Proceedings of Workshop on Applied Parallel Computing in Industrial Computation and Optimization (PARA96). Number 1184 in Lecture Notes in Computer Science, Springer-Verlag (1996) 187–195
15. Cuenca, J., Gimenez, D., Gonzalez, J.: Architecture of an automatically tuned linear algebra library. Parallel Computing **30** (2004) 187–210
16. Cuenca, J., Garcia, L.P., Gimenez, D.G.: Empirical modelling of parallel linear algebra routines. In: Proceedings of the 5th International Conference on Parallel Processing and Applied Mathematics (PPAM2003). Number 3019 in Lecture Notes in Computer Science, Springer-Verlag (2004) 169–174
17. Katagiri, T., Kuroda, H., Kanada, Y.: A methodology for automatically tuned parallel tri-diagonalization on distributed memory parallel machines. In: Proceedings of VecPar2000, Faculdade de Engenharia da Universidade do Porto, Portugal (2000) 265–277
18. Yamamoto, Y.: Performance modeling and optimal block size selection for a BLAS-3 based tridiagonalization algorithm. In: Proceedings of HPC-Asia 2005, Beijing (2005) 249–256
19. Dongarra, J., Eijkhout, V.: Self-adapting numerical software for next generation applications. International Journal of High Performance Computing Applications **17** (2003) 125–131
20. Whaley, R., Petitet, A., Dongarra, J.: Automated empirical optimizations of software and the ATLAS project. Parallel Computing **27** (2001) 3–35
21. Bilmes, J., Asanovic, K., Chin, C.W., Demmel, J.: Optimizing matrix multiply using PhiPAC: a portable, high-performance, ANSI-C coding methodology. In: Proceedings of the 11th International Conference on Supercomputing, Vienna (1997) 340–347
22. Kressner, D.: Numerical Methods for General and Structured Eigenvalue Problems. Springer-Verlag (2005)

A Parallel Mutual Information Based Image Registration Algorithm for Applications in Remote Sensing

Yunfei Du, Haifang Zhou, Panfeng Wang, Xuejun Yang, and Hengzhu Liu

School of Computer, National University of Defense Technology
Changsha, Hunan, 410073, China
forest80@163.com

Abstract. Image registration is a classical problem that addresses the problem of finding a geometric transformation that best aligns two images. Since the amount of multisensor remote sensing imagery are growing tremendously, the search for matching transformation with mutual information is very time-consuming and tedious, and fast and automatic registration of images from different sensors has become critical in the remote sensing framework. So the implementation of automatic mutual information based image registration methods on high performance machines needs to be investigated. First, this paper presents a parallel implementation of a mutual information based image registration algorithm. It takes advantage of cluster machines by partitioning of data depending on the algorithm's peculiarity. Then, the evaluation of the parallel registration method has been presented in theory and in experiments and shows that the parallel algorithm has good parallel performance and scalability.

1 Introduction

Image registration is the process by which we determine a transformation that provides the most accurate match between two or more images of approximately the same scene or objects which acquired by different sensors or taken by the same sensor but at different times. This problem often occurs in biomedical and remote sensing applications[1]. Examples of these applications include change detection using multiple images acquired at different times[2], fusion of image data from multiple sensor types (e.g., low-resolution multispectral image and high-resolution panchromatic one[3]) and medical image analysis[4].

In the remote sensing framework, the problem of registration has three peculiarities. First, remote-sensing datasets are often huge, for example, The Earth Observing System daily produces massive amounts of data approaching 1 terabyte per day[5]. Second, with the development of multiple platform remote sensing missions, multisensor satellite images often have significantly different spatial resolution when simultaneously observing the same scene. Third, image registration is computation intensive in the remote sensing applications and at the same time an image registration algorithm suitable for remote sensing applications should be as fast as possible and ideally fully automatic. Our work mostly faces to remote sensing applications, for which parallel automated image registration has become a highly desirable technique.

Some attempts to accelerate image registration by using parallel supercomputers have been done. A parallel image registration based on a global GA model was

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 464 – 473, 2006.
© Springer-Verlag Berlin Heidelberg 2006

proposed in using a parallel cluster[6]. However, the algorithm can only register monomodal remote sensing data. LeMoigne presented a fine-grain parallel algorithm for the MasPar SIMD(Single Instruction Multiple Data) architecture[7]. Substantial computational savings have been reported. Unfortunately, the parallel algorithm is not applicable to modern MIMD architectures. In our earlier work, a registration algorithm was parallelized based on a simple exhaustive search strategy[8]. Although this method is quite robust, it is not very practical for two reasons. First, exhaustive search is computationally expensive even for a small number of search parameters, and the second is it yields results of limited accuracy since accuracy depends on how fine the discrete mesh is.

The above review of parallel image registration indicates that the developed work was quite limited in scope. The rest of this paper provides a dedicated design of applicable parallel methods for registration and evaluates its performance on modern parallel cluster machine using existing remote sensing data. Section 2 describes serial image registration algorithm framework. Section 3 then describes our parallel algorithm implementation and performance analysis and associated results are presented in Section 4. Conclusions are given in Section 5. The main innovation of this paper is we address the high computation cost problem in the remote sensing mutual information based image registration applications using a parallel implementation that takes advantage of modern large-scale cluster computer architectures and the algorithm is applicable for multisensor remote sensing image registration applications, at the same time this is first study on parallel mutual information based image registration in the remote sensing framework.

2 Description of Serial Image Registration Algorithm

We describe below an intensity-based remote sensing registration algorithm, and discuss how the computational issue can be addressed using the techniques of parallel programming.

2.1 Definition and Transformation Model

First, we define image registration as follows: Given a pair of two-dimensional gray-level images, $F_R(x,y)$ and $F_T(x,y)$ that we call reference and floating image respectively with coordinates $(x,y) \in \Omega$, where Ω is a region of interest. In this paper, due to relative stability of imaging platforms and systematic data correction, global rigid transformations usually represent misalignment between satellite images quite well. The transformation can be written as

$$G_P(x,y) = T_P \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & tx \\ -\sin(\theta) & \cos(\theta) & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \quad (1)$$

where $G_P(x, y)$ is geometrical transformation, {tx, ty} are translations in x and y directions, θ is rotation angle, T_P is the transformation matrix and the vector P of translation parameters is (tx, ty, θ). To register $F_R(x,y)$ and $F_T(x,y)$ is to find the value of P

which maximizes or minimizes the similarity measure S. This can be expressed mathematically as

$$P = \arg opt(S_{T_P}(F_R, F_T)) = \arg opt(S(F_R, F_{T(T_P)})) . \quad (2)$$

2.2 Image Similarity

Similarity measure that determines how well two images are matched is a crucial part of a registration algorithm. The idea of using MI as similarity measure for registration is first introduced for medical imagery by Maes[9] and Wells[10]. Mutual information is a measure originating from information theory and the mutual information of two images is a combination of the entropy values of the images, both separately. Mutual information has been found especially robust for multisensor image registration and it has been extensive studied for the registration of medical imagery over several years[11]. In the remote sensing framework, with the rapid development of multiplatform remote sensing missions, mutual information has been applied to the registration of remote sensing imagery recently and has been a hot topic in this research area[12-14]. Our implementation of mutual information and its gradient computation is described by Maes[15].

2.3 The Analysis of Algorithm

For finding the optimal parameters that maximize the mutual information between reference and floating image, we employ conjugate-gradient methods. Our implementation closely follows the algorithm described in[16]. Conjugate-gradient method constructs the new search direction as being conjugate to the previous one with respect to the negative of mutual information to minimize.

The key steps of the algorithm are mutual information and its gradient computation between reference and floating image. Tab.1 shows the percentage of the mutual information and its gradient computation time to algorithm's total computation time. In this study, four remote sensing image datasets whose sizes are different are used in our experiments. We apply a known geometric transformation on those datasets separately. From Table.1 we can conclude mutual information and its gradient computation accounts for the bulk of the algorithm's computation cost.

These are the two aspects accounting for the bulk of the algorithm's computational cost. Computation of mutual information involves processing large amounts of data, in particular evaluating the coordinate transformation for every single pixel in the floating image to construct a joint histogram. On the other hand, the gradient computation need to construct six joint histogram derivatives and is expected to be six times as expensive as computation of mutual information itself. However, these joint histogram derivatives can be computed together with joint histogram by scanning the floating image once. So the same strategy is applied in order to parallelize the two computations.

Table 1. Percentage of the mutual information and its gradient computation time to algorithm's total computation time

	Data1	Data 2	Data3	Data4
Total time	9s	28s	113s	407s
MI and its gradient time	8.85s	27.53s	111.08s	400.59s
percentage	98.33%	98.32%	98.3%	98.28%

3 Parallel Implementation

Parallel processing has been proposed as a solution to computationally demanding problems for many years. Parallel mutual information based remote sensing image registration has been studied in this paper and implemented on a parallel cluster computer. The reasons for using cluster computer is the cluster a commercial off-the-shelf (COTS) parallel computer, is a cluster of personal computers running parallel Linux, is very easily accessible to research groups.

3.1 Data Distribution with Load Balancing

According to section 2 image registration can be decomposed into sequential processing steps, and at each step every pixel is processed in a similar manner. This makes it a good candidate for parallelization using data parallelism, rather than pipeline parallelism.

Our algorithm uses block distribution to achieve higher speedup for the hotspot of the serial algorithm. We now describe how the algorithm determines the image size for N processors. Our approach is to try to divide the floating image data into equally sized partitions for the purpose of load balancing. Each processor is assigned one of these partitions, shown as in Fig. 1.

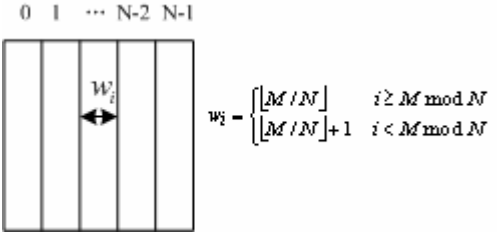


Fig. 1. Data distribution with load balancing

In detail, we assign to each processor a continuous range of columns of the floating image. When the width of the floating image M can be divided exactly by the number of processors N, each processor will be assigned the same sized partitions; When N can not divide into M, if the integer part of the width of the floating image divided by the number of processors is n, and the result of the width of the floating image modulo the number of processors is *m*, the number of columns assigned to the former *m* processors is (*n*+1) and assigned to every latter processor is *n*.

3.2 Parallel Mutual Information and Its Gradient Computation

Floating image transformation is a local operation and each sample computation is independent of others. Each node computes transformed coordinates for assigned partition of floating image according to transformation matrix.

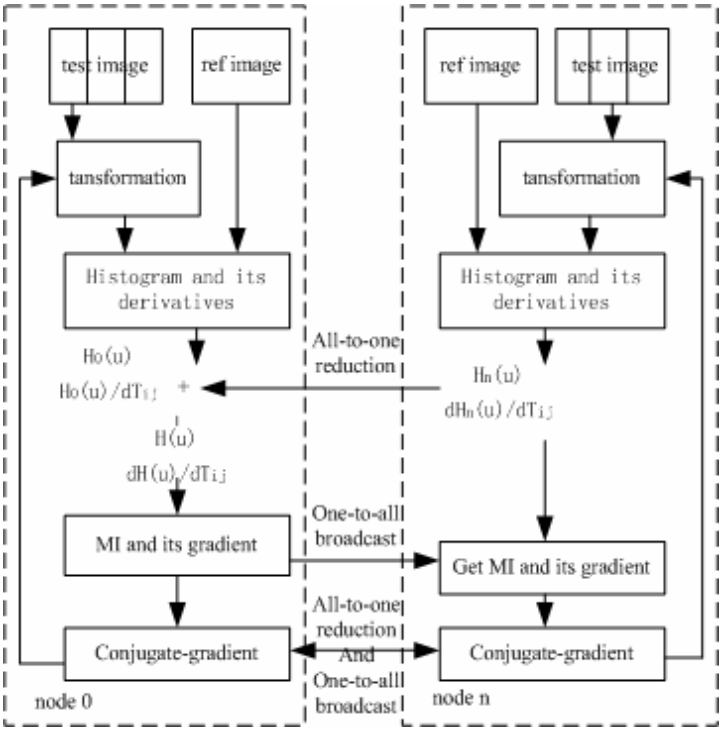


Fig. 2. Parallel registration algorithm framework

Computation of mutual information and its gradient is a global operation that involves the whole image. The pixel pairs encountered by each node are stored in separate 2-D histogram. After finishing its contribution to the mutual information and its gradient, each node performs all-to-one reduction communication to get global histogram and its derivatives with respect to transformation matrix T. A naive way to get global result is to sequentially send one message from the source to the destination process and then the data from all processes are combined through an associative operator and accumulated at the destination process. However, this is inefficient because the destination process becomes a bottleneck. A better reduction algorithm can be devised using a technique commonly known as recursive doubling[17].

According to section 2 the main computation of conjugate gradient optimization method in our serial registration algorithm is mutual information computation. To parallelize conjugate gradient optimization is to parallelize the computation of mutual information which has described above.

A flowchart of the parallel registration algorithm in this paper is shown in Fig. 2. Obviously, the load is almost balanced. However, extra communication is introduced into global operation for computing the mutual information and its gradient over the same image.

3.3 Run Time Analysis

Here we do quantitative evaluation for our parallel algorithm and give the complexity analysis. LogGP is used as a computation model for analyzing our parallel algorithm on distributed memory machines[18]. The computation cost of algorithm will be evaluated in terms of two measures: the computation time T_{comp} and the communication time T_{comm}. For communication time, let t_α is the startup time for a message, and t_β is transfer time per data of double type. Original image size is M*M, the number of histogram bins is K in the computation and the parallel system has N processors. There are R solutions at each line minimization for optimum along conjugate gradient directions. The cost of joint histogram and its derivatives is $O(M^2)$ and the computational cost relative to the number of histogram bins K used in the computation is $O(K^2)$. So the serial execution time is $T(1) = O(M^2+K^2)$.

In our parallel algorithm the maximum number of pixels assigned to each processor to compute mutual information and its gradient is $(\lfloor M/N \rfloor+1)\cdot M$, so the computation cost is

$$T_{comp} = O((\left\lfloor \frac{M}{N} \right\rfloor+1)\cdot M + K^2) = O(\frac{M^2 - M^2 \% N}{N} + M + K^2) \quad (3)$$

When computing mutual information and its gradient there are two communication steps, one of which is reduction and another is broadcast. The transferred data in the reduction operation contains a histogram and six histogram derivatives dH/dT_{ij}, and each histogram has K^2 data, so the communicational cost is $T_{comm1} = O(t_\alpha + t_\beta \cdot 7K^2\lceil \lg N \rceil)$. The transferred data in the broadcast operation is mutual information and its gradient values, so the communicational cost is $T_{comm2} = O(t_\alpha + t_\beta \cdot 4\lceil \lg N \rceil)$.

When performing conjugate gradient optimization there are also two communication steps, one of which is reduction and another is broadcast. The transferred data in the reduction operation contains a histogram which has K^2 data, so the communicational cost is $T_{comm3} = O((t_\alpha + t_\beta \cdot K^2\lceil \lg N \rceil)\cdot R)$. The transferred data in the broadcast operation is mutual information, so the communicational cost is $T_{comm4} = O((t_\alpha + t_\beta \cdot \lceil \lg N \rceil)\cdot R)$.

So the communicational cost in our parallel algorithm is

$$T_{comm} = T_{comm1} + T_{comm2} + T_{comm3} + T_{comm4} = O(t_\alpha \cdot R + t_\beta \cdot (R+7)K^2\lceil \lg N \rceil) \quad (4)$$

In sum the overall parallel execution time is

$$\begin{aligned} T(n) &= T_{comp} + T_{comm} \\ &= O(\frac{M^2 - M^2 \% N}{N} + M + K^2 + t_\alpha \cdot R + t_\beta \cdot (R+7)K^2\lceil \lg N \rceil) \end{aligned} \quad (5)$$

The increase in speed due to parallelizing an algorithm with N processors is evaluated by Amdahl's Law[21] and is given by following expression

$$\begin{aligned} S &= \frac{T(1)}{T(n)} = \frac{O(M^2 + K^2)}{O(\frac{M^2 - M^2 \% N}{N} + M + K^2 + t_\alpha \cdot R + t_\beta \cdot (R+7)K^2\lceil \lg N \rceil)} \\ &\approx O(\frac{N}{1}) = O(N) \quad M \to \infty \end{aligned} \quad (6)$$

The efficiency of the parallel program can be written as

$$E = \frac{S}{N} = O(\frac{(1+K^2/M^2)}{1-\frac{M^2\%N}{M^2}+\frac{N}{M}+\frac{K^2\cdot N}{M^2}+\frac{K^2}{M^2}+\frac{t_\alpha\cdot R+t_\beta\cdot(R+7)K^2\lceil\lg N\rceil}{M^2}}) \approx O(1) \quad M\to\infty \tag{7}$$

According to expression (6) and (7) with the increase of image size, the speedup is approaching N and efficiency is approaching 1, and this shows that our parallel algorithm has good parallel performance and scalability.

4 Application Results

In this section illustrative results of the parallel registration algorithm with four pairs of remote sensing images are presented. The four pairs are images used in Section II. Our parallel algorithm is coded entirely in C in which communication is handled with the message passing standard MPI, blocking message passing routines is used and compiled using version 2.0 of mpi compiler suite[19].

The experiment was performed on a cluster computer with 16 nodes with dual Intel Xeon 3.4GHz CPU, 4G RAM, and a 60GB SCSI disk drive running Linux 2.4.20. All nodes are connected through a 1Gbps Ethernet LAN.

Fig.3 shows the registration result of a pair of test images, in which Fig.3(c) is the output of the transformed floating image matched with reference image by mapping function.

(a) Reference image (b) Floating image (c) Output image after registration

Fig. 3. Registration result of test images

Fig.4 presents the computation speedups and Fig.5 presents the efficiencies achieved by different datasets on our parallel platform. Fig.4 shows that with the increase of image size the speedup of our algorithm goes up rapidly when the computation to communication ratio is high enough to get benefit from load balance. However, when the number of processors is increased to a special scale for given image size the speedup of the algorithm decreases because the computation to communication ratio goes down. Fig.5 shows that for a given image size, when the number of processors is increased, the overall efficiency of the parallel algorithm goes down. In most cases, the efficiency of the parallel algorithm increases if the image size is increased while keeping the number of processors constant. In sum the experiment about the performance characteristics of the parallel algorithm is consistent with theoretical analysis and shows that our parallel algorithm has good parallel performance and scalability.

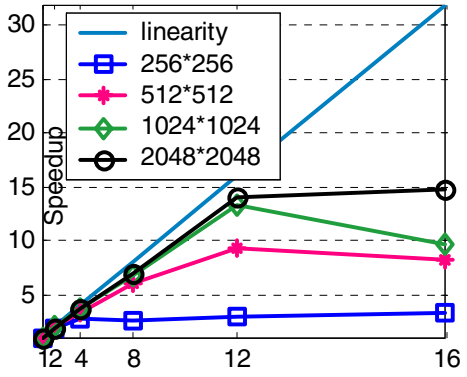


Fig. 4. Speedups of the algorithm with different datasets

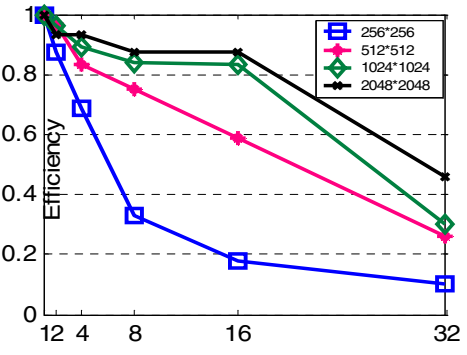


Fig. 5. Efficiencies of the algorithm with different datasets

5 Conclusions

Image registration through maximization of mutual information is a hot topic in the area of multisensor image registration and computationally intensive. Since both the need for fast and reliable image registration and the amount of multisensor remote sensing imagery are growing tremendously, the implementation of automatic image registration methods on high performance machines needs to be investigated. Based on analyses of existing serial image registration, a parallel algorithm based on data parallelism for real-time mutual information based image registration is developed. Then, the evaluation of the parallel registration method has been presented in theory and in experiments.

Future work will include the study of parallel multiresolution mutual information based image registration. Nonrigid image registration is a more complex problem, and parallel nonrigid image registration based on different transformation models will be our research topic and the papers about them will be published later.

Acknowledgement

The authors would like to thank Frederik Maes for his generous help and expert advice on mutual information and its gradient computation, and Philippe Thévenaz for his assistance with serial image registration algorithm. The authors would also like to acknowledge the support of high performance grid application based on IPv6 project sponsored by National Development and Reform Commission under the grant CNGI-04-15-7A.

References

1. Lisa Gottesfeld Brown,: A survey of image registration techniques. ACM Comput. Surv., vol.24, no.4 (1992) 325-376
2. J. R. G. Townshend, C. O. Justice, C. Gurney, and J. McManus.: The impact of misregistration on change detection. IEEE Trans. Geosci. and Remote Sensing, vol. 30 (1992) 1054–1060.
3. X. D. S. Khorram.: A hierarchical methodology framework for multisource data fusion in vegetation classification. Int. J. Remote Sens., vol. 19, no. 18 (1998) 3697–3701.
4. J.B.Antoine Maintz, Max A. Viergever.: A survey of medical image registration. Medical Image Analysis, vol.2, no.1 (1998) 1-36
5. M. Kafatos, L. Bergman, R. Chinman, T. El-Ghazawi, S. Nittel, L. Olsen, X. S. Wang.: Data exchanges and interoperability in distributed Earth science information systems. Proceedings of 11th International Conference on Scientific and Statistical Database Management, Cleveland, Ohio, USA (1999)
6. Prachya Chalermwat, Tarek El-Ghazawi, Jacqueline LeMoigne.: GA-based Parallel Image Registration on Parallel Clusters. IPPS/SPDP Workshops (1999)
7. Jacqueline Le Moigne, William J. Campbell, Robert F. Cromp.: An automated Parallel Image Registration Technique Based on the Correlation of Wavelet Features. IEEE Trans. Geosci. and Remote Sensing, vol. 40, no. 8 (2002) 1849-1864
8. Haifang Zhou, Xuejun Yang, Hengzhu Liu, Yu Tang.: First Evaluation of Parallel Methods of Automatic Global Image Registration Based on Wavelets. The International Conference on Parallel Processing, Oslo, Norway (2005) 129-136
9. Frederik Maes, Andre Collignon, Dirk Vandermeulen, Guy Marchal, Paul Suetens.: Multimodality Image Registration by Maximization of Mutual Information. IEEE Trans. Medical imaging, vol.16, no.2 (1997) 187-198
10. W. M.Wells III, P. Viola, R. Kikinis.: Multi-modal Volume Registration by Maximization of Mutual Information. Medical Robotics and Computer Assisted Surgery, John Wiley & Sons, New York (1995) 55-62
11. Josien P.W. Pluim, J.B. Antoine Maintz, Max A. Viergever.: Mutual Information Based Registration of Medical Images: a Survey. IEEE Trans. Medical Imaging, Vol.22, No.8 (2003) 986-1004
12. K. Johnson, A. Cole-Rhodes, I. Zavorin, J. Le Moigne.: Mutual information as a similarity measure for remote sensing image registration. Proc. SPIE Aerosense 2001, Geo-Spatial Image and Data Exploitation II, vol. 4383, Orlando, FL (2001) 51–61
13. Hua-Mei Chen, Pramod K. Varshney, Manoj K. Arora.: Performance of Mutual Information Similarity Measure for Registration of Multitemporal Remote Sensing Images. IEEE Trans. Geosci. and Remote Sensing, vol.41, no.11 (2003) 2445-2454

14. Arlene A. Cole-Rhodes, Kisha L. Johnson, Jacqueline LeMoigne, Ilya Zavorin.: Multiresolution Registration of Remote Sensing Imagery by Optimization of Mutual Information Using a Stochastic Gradient. IEEE Trans. Image Processing, vol.12, no.12 (2003) 1495-1511
15. F. Maes, D. Vandermeulen, P. Suetens.: Comparative evaluation of multiresolution optimization strategies for multimodality image registration by maximization of mutual information. Medical Image Analysis, vol. 3, no. 4 (1999) 373–386
16. W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery.: Numerical Recipes in C. Cambridge Univ. Press (1992)
17. A. Alexandrov, M. Ionescu, K.E. Schauser, C. Scheiman.: LogGP: incorporating long messages into the LogP model - one step closer towards a realistic model of parallel computation. Procs. of the 7th Annual ACM Symp. on Parallel Algorithms and Architectures (1995) 95-105
18. G. M. Amdahl.: Validity of the single-processor approach to achieving large scale computing capabilities. Proc. AFIPS Conf., vol. 30, Reston, VA (1967) 483–485
19. http://www.mcs.anl.gov/mpi/mpich2

A Novel Broadcasting Algorithm for Wireless Sensor Networks

Yong Tang and Mingtian Zhou

College of Computing Science and Engineering,
University of Electronic Science and Technology of China,
Chengdu, Sichuan, China
worldgulit@tom.com, mtzhou@uestc.edu.cn

Abstract. Broadcasting is one of the typical operations in wireless sensor networks where the nodes are energy constrained. Considering the characteristics that nodes are almost static and position aware in wireless sensor networks, we propose a novel broadcasting algorithm (NBA) for utilizing energy efficiently. The energy status of nodes is regarded as an important factor, as well as the local network topology, in clustering and deciding retransmission nodes. We prove the accurateness of NBA. Simulations show that NBA reduces the number of redundant retransmissions and balances traffic efficiently, so energy is conserved and energy dissipation is distributed evenly throughout the network. Thereby the lifetime of WSN is prolonged greatly.

1 Introduction

Recent technological advances have enabled large scale information gathering through a network of tiny, low power sensors. This network of sensor nodes, known as wireless sensor networks (WSN), has revolutionized remote monitoring applications because of its ease of deployment, ad hoc connectivity and cost-effectiveness [1, 2].

Broadcasting is one of the typical operations in WSN [3]. For instance, broadcasting can be used to explore routing paths if the topology of network is unknown beforehand and advertise important messages (e.g., an intruder detected by a sensor) to all nodes. The straightforward solution for broadcasting in WSN is flooding (blind flooding), however blind flooding is improper in WSN since it consumes too much energy which is deeply constrained in WSN [4, 5, 6]. Furthermore, as a node has a limited battery power and data transmission dominates energy consumption of a node when compared with sensing and computation operations [1, 2, 5, 7], broadcasting algorithms designed for traditional MANET are not suitable for WSN.

In WSN, the network lifetime can be measured with respect to the time until a certain percentage of nodes in the network are energy-depleted [8], so energy must be conserved and energy dissipation must be distributed evenly throughout the nodes. Therefore in WSN broadcasting algorithms, balancing traffic is a key factor to maximize the network lifetime as well as reducing redundant transmission nodes.

Considering the characteristics of WSN that nodes almost are static and energy constrained, we proposed a novel broadcasting algorithm for WSN (we call this algorithm as NBA), which do clustering first and then deciding transmission nodes

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 474 – 484, 2006.
© Springer-Verlag Berlin Heidelberg 2006

in each round [9]. This scheme reduces transmission nodes greatly and balances traffic well under the condition that all the nodes in WSN receive the advertised message.

To evaluate the performance of our proposed algorithm, we simulate NBA and some good broadcasting algorithms for MANET including MPR [10] and EMPR [11]. The simulation shows that NBA greatly outperforms MPR and EMPR in terms of reducing transmitting nodes and balancing traffic. So NBA can prolong the network lifetime greatly in broadcasting of WSN.

2 Preliminaries of NBA

In this paper, our model WSN has the following properties:

(1) The WSN is connected.

(2) The nodes in WSN have the same communication radius, and have the omnidirectional antennas to transmit by the same rated power.

(3) The nodes in WSN are aware of their position which identifies them uniquely.

(4) The nodes in WSN are static.

(5) If the distance between nodes in WSN is not more than communication radius, the nodes can communicate correctly through local wireless broadcasting channel.

NBA consists of two stages. One is clustering and the other is transmission deciding. Clustering algorithm elects cluster heads which have more residual energy and stronger connectivity than its neighbors. Then all the nodes are divided into three types: cluster head (CH), member affiliated to multiple clusters (MM) and member affiliated to single cluster (SM). The nodes of different types have some different properties. CH nodes are not adjacent one another, SM nodes are adjacent to one and only one certain CH node, and MM nodes are adjacent to at least two certain CH nodes. So if all the CH nodes do transmissions in WSN, all the other nodes can receive the advertised message. Moreover, as nodes have more residual energy and stronger connectivity than other nodes, energy is conserved and traffic is balanced via periodical clustering and transmission deciding in WSN. But as CH nodes are not adjacent one another, SM/MM nodes have to do transmissions to bridge all the CH nodes. Based on this idea, we design the transmission deciding algorithm of NBA. Transmission deciding algorithm allows almost all the CH nodes to do transmissions in WSN except some particular CH nodes, and tries to choose SM/MM nodes which have more residual energy and stronger connectivity to do transmissions. So broadcasting can be completed correctly in WSN, and energy is conserved and traffic is balanced as well.

Some common notations and definitions are given for describing NBA:

(1) $class(u)$ represents the type of node u.

(2) V represents all the nodes in WSN. V_{ch} represents all the CH nodes determined by clustering. V_{mm} represents all the MM nodes determined by clustering. V_{sm} represents all the SM nodes determined by clustering.

(3) $p(u)=(x(u),y(u))$ represents the position of node u, where $x(u)$ and $y(u)$ is X and Y coordinates of u.

(4) $\overline{d(u)} = \sqrt{x(u)^2 + y(u)^2}$.

(5) $p(u)>p(v)$ represents $((\overline{d(u)} > \overline{d(v)}) \vee ((\overline{d(u)} = \overline{d(v)}) \wedge (|x(u)|>|x(v)|)))$.

(6) If the distance between two nodes is less than or equal to communication radius, we call these two nodes are adjacent, and neighbors of u are represent as $N(u)$.

(7) $d(u)=|N(u)|$, where $|N(u)|$ represents the size of $N(u)$.

(8) $N_{ch}(u)=\{x|(x\in N(u)) \wedge (class(x)=\text{CH})\}$.

(9) $N_{mm}(u)=\{x|(x\in N(u)) \wedge (class(x)=\text{MM})\}$.

(10) $N_{sm}(u)=\{x|(x\in N(u)) \wedge (class(x)=\text{SM})\}$.

3 Clustering Algorithm

The following control message will be used in clustering algorithm:

(1) *HELLO*: Node advertises its existence to neighbors.
(2) *ADV*: Node advertises itself a CH node.
(3) *ABD*: Node advertises itself not a CH node.
(4) *IFM*: Node advertises information to neighbors.

The following symbols will be used in clustering algorithm:

(1) $weight(u) = d(u) \times (1 + \lambda \times residual(u) / ini(u))$, where $residual(u)$ and $ini(u)$ represent the residual energy and the initial energy of u respectively, λ ($\lambda \geq 0$) is an energy aware parameter.

(2) $N_{abd}(u)$ represents the neighbors of u which have sent control message *ABD*.

3.1 Description of Clustering Algrithm

When initializing, clustering algorithm assigns $class(x) \leftarrow$ null, $N_{ch}(x) \leftarrow \Phi$, $N_{mm}(x) \leftarrow \Phi$ and $N_{abd}(x) \leftarrow \Phi$ for every node x ($x \in V$). Then clustering algorithm is divided into three phases: neighbor finding, electing and information exchanging.

3.1.1 Neighbor Finding

Every node x ($x \in V$) sends $HELLO(p(x))$ via local wireless channel. According to assumptions 2, 3, 4 and 5, x can find their neighbors. Then x sends $IFM(p(x),weight(x))$ locally, and every node y ($y \in N(x)$) stores $weight(x)$.

At this phase, the message complexity of each node is $O(2)$.

3.1.2 Electing

In this phase, all the nodes check their types and prepare to receive *ADV* and *ABD*, and then:

(1) When the type of node u ($u \in V$) is null, i.e. $class(u)$=null

u assigns $class(u) \leftarrow$ CH and $N_{ch}(u) \leftarrow \Phi$. Then u sends $ADV(p(u))$ locally to advertise u a CH node if every node x ($x \in N(u)$) satisfies $(weight(u)>weight(x)) \vee ((weight(u)=weight(x)) \wedge (p(u)<p(x)))$.

(2) When node v ($v \in N(u)$) receives $ADV(p(u))$

v assigns $N_{ch}(v) \leftarrow (N_{ch}(v) \cup \{u\})$. Then v assigns $class(v) \leftarrow$SM if $|N_{ch}(v)|=1$, otherwise $class(v) \leftarrow$MM. And then v sends $ABD(p(v))$ locally to advertise v not a CH node if it receives ADV at the first time.

(3) When node w ($w \in N(v)$) receives $ABD(p(v))$

w assigns $N_{abd}(w) \leftarrow (N_{abd}(w) \cup \{v\})$ if $class(w)$=null. Then w assigns $class(w) \leftarrow$CH and $N_{ch}(w) \leftarrow \Phi$ and w sends $ADV(p(w))$ locally to advertise w a CH node if every node x ($x \in (N(w) - N_{abd}(w))$) satisfies $(weight(w) > weight(x)) \vee ((weight(w) = weight(x)) \wedge (p(w) < p(x)))$.

At this phase, the message complexity of each node is $O(1)$.

3.1.3 Information Exchanging

Every node x ($x \in V_{mm}$) sends $IFM(p(x), class(x))$ locally to advertise x a MM node, and every node y ($y \in N(x)$) assigns $N_{mm}(y) \leftarrow (N_{mm}(y) \cup \{x\})$ when y receives IFM. Then x sends $IFM(p(x), class(x), N_{ch}(x)$ and $N_{mm}(x))$ locally, and y saves $class(x)$, $N_{ch}(x)$ and $N_{mm}(x)$ when y receives IFM.

At this phase, the message complexity of each CH/SM node is $O(1)$ and that of each MM node is $O(2)$.

It is obvious that the message complexity of each node in clustering algorithm is $O(5)$ in the worst case.

3.2 Properties of Nodes After Clustering

The nodes in WSN have the following properties apparently after clustering:

Property 1: Sets V_{ch}, V_{mm} and V_{sm} are the partition of set V.

Property 2: For every node x ($x \in V_{sm}$) there exists one and only one node u ($u \in N(x)$) which satisfies $class(u)$=CH. For every node x ($x \in V_{mm}$) there exist n($n \geq 2$) nodes $u_1, u_2 \ldots u_i \ldots u_n$ ($u_1, u_2 \ldots u_i \ldots u_n \in N(x)$) which satisfy $class(u_1)$=CH, $class(u_2)$=CH…$class(u_i)$=CH…$class(u_n)$=CH respectively. For every node x ($x \in V_{ch}$) there exists no node u ($u \in N(x)$) which satisfies $class(u)$=CH.

4 Transmission Deciding Algorithm

Transmission deciding algorithm executed after clustering algorithm. The following control messages will be used in transmission deciding:

(1) $APP0$: MM node advertises itself a transmission node.
(2) $APP1$: CH node appoint MM node as a transmission node.
(3) $APP2$: SM node appoint MM node as a transmission node.

The following symbols will be used in transmission deciding:

(1) The value of Boolean function $transmission(u)$ is TRUE if node u does transmission, otherwise $transmission(u)$ is FALSE.
(2) $RTM_{mm}(u)$ represents MM neighbors of u which do transmissions.
(3) $RTM_{ch}(u)=\{x | (x \in \bigcup_{(y \in N(u))} N_{ch}(y)) \wedge (p(x) < p(u))\}$, where $class(u)$=CH.

(4) There is an equation $\{ch(u)\}= N_{ch}(u)$ if $class(u)$=SM. That is to say, $ch(u)$ is the unique CH node which is adjacent to u.

(5) $RTM_{ch\text{-}2}(u)=\{x|(x\in \bigcup_{(y\in N_{mm}(u)\wedge y\notin N_{ch}(ch(u)))} N_{ch}(y))\}$, where $class(u)$=SM.

(6) $RTM_{ch}(v,u)=\{x|(x\in RTM_{ch}(u))\wedge(x\in N(v))\}$, where $v\in N(u)$ and $class(v)$=MM.

(7) $RTM_{ch\text{-}2}(v,u)=\{x|(x\in RTM_{ch\text{-}2}(u))\wedge(x\in N(v))\}$, where $v\in N(u)$ and $class(v)$=MM.

(8) Boolean function *chosen* and symbol *CHOSEN* are introduced for convenience, and their meanings are described in the algorithm below.

(9) $D(v,u)=\{x|(x\in RTM_{ch}(v,u))\wedge(chosen(x)=\text{FALSE})\}$.

(10) $D2(v,u)=\{x|(x\in RTM_{ch\text{-}2}(v,u))\wedge(chosen(x)=\text{FALSE})\}$.

(11) $WEIGHT(De(v,u))=|De(v,u)|\times(1+\lambda\times residual(v)/ini(v))$, where the means of $residual(v)$, $ini(v)$ and λ are same as mentioned above. $De(v,u)$ represents $D(v,u)$ or $D2(v,u)$.

(12) When the transmission deciding algorithm finished, R_{sm} represents SM nodes which do transmissions, R_{mm} represents MM nodes which do transmissions, and R_{ch} represents CH nodes which do transmissions.

4.1 Description of Transmission Deciding Algrithm

When initializing, transmission deciding algorithm assigns $transmission(x)\leftarrow$TRUE and $RTM_{mm}(x)\leftarrow\Phi$ for every node x ($x\in V_{ch}$), and assigns $transmission(y)\leftarrow$FALSE and $RTM_{mm}(y)\leftarrow\Phi$ for every node y ($y\notin V_{ch}$). Then the following three sub-algorithms are executed to determine the transmission nodes:

Sub-algorithm 1 determines whether SM nodes to do transmissions.

Every node u ($u\in V_{sm}$) assigns $transmission(u)\leftarrow$TRUE if the network consisting of $(\bigcup_{x\in N(u)} N_{ch}(x))\cup N_{ch}(u)\cup N_{mm}(u)$ $(\bigcup_{x\in N(u)} N_{mm}(x))$ is separated.

Sub-algorithm 2 determines whether MM nodes to do transmissions. Sub-algorithm 2 includes sub-algorithm 2.1, sub-algorithm 2.2 and sub-algorithm 2.3.

Sub-algorithm 2.1

Every node u ($u\in V_{mm}$) assigns $transmission(u)\leftarrow$TRUE and sends $APP0(p(u))$ locally if there exists one node v ($v\in N_{mm}(u)$) which satisfies $N_{ch}(u)\cap N_{ch}(v)=\Phi$. Every node x ($x\in(N(u)\cap(N_{ch}(u)\cup N_{sm}(u))))$ assigns $RTM_{mm}(x)\leftarrow(RTM_{mm}(x)\cup\{u\})$ when x receives $APP0$.

Sub-algorithm 2.2

Every node u ($u\in V_{ch}$) appoints MM nodes to do transmissions according to the following steps:

(a) u assigns $CHOSEN(u)\leftarrow\Phi$.

(b) u assigns $chosen(x)\leftarrow$TRUE for every node x $(x\in(RTM_{ch}(u)\cap(\bigcup_{(y\in RTM_{mm}(u))} N_{ch}(y))))$ and $chosen(x)\leftarrow$FALSE for every node x $(x\in(RTM_{ch}(u)-\bigcup_{(y\in RTM_{mm}(u))} N_{ch}(y)))$.

(c) u assigns $CHOSEN(u)\leftarrow(CHOSEN(u)\cup\{v\})$ and $chosen(x)\leftarrow$TRUE for every node x ($x\in D(v,u)$) if node v satisfies $p(v)=\min\{p(x)|WEIGHT(D(x,u))\in\{\max\{WEIGHT(D(y,u))|y\in N_{mm}(u)\}\}\}$.

(d) If $\{x|(x\in RTM_{ch}(u))\wedge(chosen(x)=\text{FALSE})\}\neq\Phi$, then goto (c).

(e) u sends $APP1(CHOSEN(u))$ locally. Every node x ($x\in CHOSEN(u)$) assigns $transmission(x)\leftarrow$TRUE when x receives $APP1$, and every node y ($y\in N_{sm}(u)$) assigns $RTM_{mm}(y)\leftarrow(RTM_{mm}(y)\cup CHOSEN(u))$ when y receives $APP1$.

Sub-algorithm 2.3

Every node u ($u\in V_{sm}$) appoints MM nodes to do transmissions according the steps similar to sub-algorithm 2.2 if $transmission(u)$=TRUE:

(a) u assigns $CHOSEN(u)\leftarrow\Phi$.

(b) u assigns $chosen(x)\leftarrow$TRUE for every node x ($x\in(RTM_{ch\text{-}2}(u)\cap(\bigcup_{(y\in RTM_{mm}(u))}N_{ch}(y))$) and $chosen(x)\leftarrow$FALSE for every node x ($x\in(RTM_{ch}(u)-\bigcup_{(y\in RTM_{mm}(u))}N_{ch}(y))$).

(c) u assigns $CHOSEN(u)\leftarrow(CHOSEN(u)\cup\{v\})$ and $chosen(x)\leftarrow$TRUE for every node x ($x\in D2(v,\underline{u})$) if node v satisfies $p(v)=\min\{p(x)|WEIGHT(D2(x,u))\in\{\max\{WEIGHT(D2(y,u))|y\in N_{mm}(u)\}\}\}$.

(d) If $\{x|(x\in RTM_{ch\text{-}2}(u))\wedge(chosen(x)=\text{FALSE})\}\neq\Phi$, then goto (c).

(e) u sends $APP2(CHOSEN(u))$ locally. Every node x ($x\in CHOSEN(u)$) assigns $transmission(x)\leftarrow$TRUE when x receives $APP2$.

Sub-algorithm 3 determines whether CH nodes to do transmissions.

Every node u ($u\in V_{ch}$) assigns $transmission(u)\leftarrow$FALSE if $d(u)=1$.

It is obvious that the message complexity of each node in transmission deciding algorithm is $O(1)$ in the worst case.

4.2 Proof of Accurateness of NBA

If all the nodes in WSN can receive the advertised message, that is to say, the one-to-all semantics of broadcasting can be guaranteed, the broadcasting algorithm is accurate.

In transmission deciding algorithm, sub-algorithm 2.3 depends on sub-algorithm 1, 2.1 and 2.2, and sub-algorithm 2.2 depends on sub-algorithm 2.1, and the other sub-algorithms only depend on local information, so sub-algorithms 1, 2.1 and 3 can run concurrently. However, for analyzing the accurateness of transmission deciding algorithm conveniently, we assume that sub-algorithm 1, 2.1, 2.2, 2.3 and 3 run in sequence. Then we explain the sub-algorithms respectively in detail as follows.

In sub-algorithm 1, SM node u does transmission only when it satisfies the proposition that the network consisting of CH and MM nodes which are 1- or 2-hops away from u is separated (we call it proposition 1). In fact, proposition 1 is a necessary condition for another proposition that the network is separated if node u removes from the network (we call it proposition 2). Obviously, proposition 1 is a necessary condition of u being a cut point and proposition 2 is a sufficient and necessary condition of u being a cut point in Graph Theory. So SM node which does no transmission must not be a cut point, and SM nodes which do no transmissions must not be point cut because none of them is used to judge proposition 1. Thereby the network consisting of R_{sm}, V_{ch} and V_{mm} is connected, and broadcasting performs accurately after sub-algorithm 1.

If V_{sm} is removed, WSN are divided into multiple connected sub-networks, called CH-MM sub-networks, which only consist of CH and MM nodes. And all the CH-MM sub-networks and R_{sm} constitute a connected network. Sub-algorithm 2.1 and 2.2 reduce MM nodes under the precondition that the advertised message can be transmitted among CH nodes of a CH-MM sub-network. Sub-algorithm 2.3 reduces MM nodes under the precondition that the advertised message can be transmitted among CH nodes of CH-MM sub-networks via nodes in R_{sm}. Then we analyze Sub-algorithm 2.1, 2.2 and 2.3 in detail as follows.

According to the properties of nodes after clustering, two CH nodes in a CH-MM sub-network can reach each other by combinations of two types of basic path appropriately, which are called as P1 and P2 respectively:

(1) P1 represents the type of path through which two CH nodes can reach each other only by MM node sequence. Assume the type of path P is P1 and $P=c_1m_1m_2\ldots m_im_{i+1}\ldots m_nc_2$, where m_i is an MM node ($1\leq i\leq n$ and $n\geq 2$), and c_1 and c_2 are both CH nodes. m_i and m_{i+1} must satisfy $N_{ch}(m_i)\cap N_{ch}(m_{i+1})=\Phi$, otherwise m_i and m_{i+1} can reach each other by their mutual CH neighbor and we can get the contradictive conclusion that the type of path P is not P1. So m_i and m_{i+1} do transmissions in sub-algorithm 2.1. Therefore, CH nodes of a CH-MM sub-network which reach each other via P1 still remain reachable after sub-algorithm 2.1.

(2) P2 represents the type of path through which two CH nodes can reach each other only by alternate sequence between MM and CH node. Assume the type of path P is P2, and $P=c_1m_1\ldots c_im_ic_{i+1}m_{i+1}\ldots c_nm_nc_{n+1}$, where $c_i(1\leq i\leq n+1)$ is a CH node and $m_i(1\leq i\leq n)$ is a MM node. As c_i and c_{i+1} satisfy $(p(c_i)> p(c_{i+1})\vee p(c_{i+1})> p(c_i))$, c_i(or c_{i+1}) will appoint a mutual MM neighbor to do transmission in sub-algorithm 2.2. Therefore, CH nodes of a CH-MM sub-network which reach each other via P2 still remain reachable after sub-algorithm 2.2.

Obviously, if there are paths which are combinations of P1 and P2 between two CH nodes of a CH-MM sub-network, these two CH nodes still remain reachable after sub-algorithm 2.1 and 2.2.

In sub-algorithm 2.3, it is a necessary condition that MM neighbors of SM node u are not neighbors of $N_{ch}(u)$ for the situation that CH neighbors of MM neighbor of u are not in the CH-MM sub-network which includes $N_{ch}(u)$. So u in R_{sm} appoints MM nodes to do transmissions to assure the advertised message can be transmitted between these CH nodes and $N_{ch}(u)$.

In sub-algorithm 3, CH node does no transmission only when it is a leaf node. It is obvious that broadcasting can perform accurately after sub-algorithm 3.

In a word, broadcasting can perform accurately by doing transmissions via nodes in R_{sm}, R_{mm} and R_{ch} after sub-algorithm 1, 2 and 3. So we can conclude that NBA is accurate.

5 Performance Evaluation

In order to evaluate the performance of NBA, MPR EMPR and NBA are compared in terms of average residual energy (ARE), average energy bias (AEB), node fault ratio (NFR) and minimum residual energy (MRE). ARE, AEB, NFR and MRE are defined as follows:

$$ARE = \frac{\sum_{x \in V} residual(x)}{|V|}. \tag{1}$$

$$AEB = \frac{\sum_{x \in V} |residual(x) - ARE|}{|V|}. \tag{2}$$

$$NFR = \frac{\sum_{x \in V} isFault(x)}{|V|}. \tag{3}$$

$$MRE = \min\{residual(x) \mid x \in V\}. \tag{4}$$

Where in (3) $isFault(x)=1$ if and only if $residual(x)=0$, otherwise $isFault(x)=0$.

To analyze energy dissipation of broadcasting algorithms conveniently, we assume that nodes consume energy only when they do transmissions for broadcasting. The more ARE means that there are less redundant transmissions. The less AEB means that traffic is balanced better. The less NFR means that the network lifetime is much longer. The more MRE means that the network will survive longer. ARE and AEB indicate the lifetime of WSN indirectly; NFR and MRE indicate the lifetime of WSN directly.

5.1 Simulation Model

Assume that the initial energy of each node in WSN is 1J. Broadcasting performs in rounds, and many messages can be advertised in each round. Each node doing transmission consumes 0.01J of energy in each round, otherwise nodes consume no energy.

We simulate MPR, EMPR and NBA, and in NBA $\lambda=0$, $\lambda=0.5$, $\lambda=1$ and $\lambda=5$ respectively. The experiment is described as follows.

The communication radius of node is 50m. 100 nodes are uniformly deployed at random in two dimensioned planes: $50\times300m^2$, $100\times300m^2$, and $150\times300m^2$ to generate a connected network respectively. After the energy of each node is initialized, broadcasting performs 100 times for each algorithm in each network. Then we compute ARE, AEB, NFR and MRE. The experiment is repeated for 100 times, and the average result is regarded as the final result.

5.2 Simulation Results

The differences in ARE, AEB, NFR and MRE of MPR, EMPR and NBA ($\lambda=0$, $\lambda=0.5$, $\lambda=1$, $\lambda=5$) are shown in Fig. 1, Fig. 2, Fig. 3 and Fig. 4. When NFR or MRE is not shown in some figures, it means that the value of it is zero under the corresponding experimental conditions.

We can see that from these four figures:

ARE of NBA ($\lambda=0$, $\lambda=0.5$, $\lambda=1$, $\lambda=5$) is far more than that of MPR and EMPR, and AER of NBA ($\lambda=0$, $\lambda=0.5$, $\lambda=1$, $\lambda=5$) is far less than that of MPR and EMPR.

ARE of NBA ($\lambda=0$) is the most in that of all algorithms, and AER of NBA ($\lambda=0$) is more than NBA ($\lambda\neq0$). The reason is that NBA ($\lambda=0$) only tries to reduce

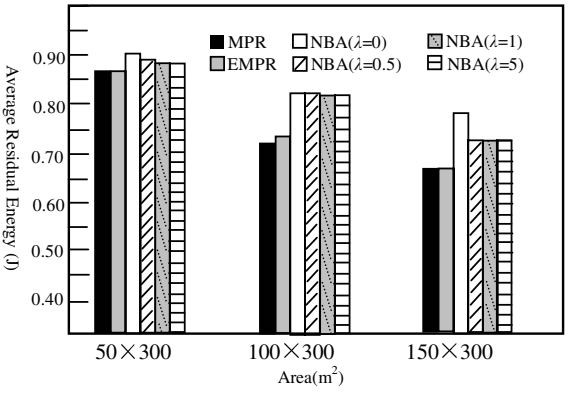


Fig. 1. ARE after 100 round broadcasting, when node number is 100, different areas

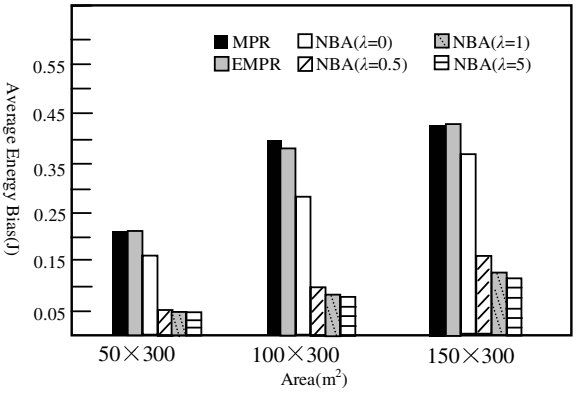


Fig. 2. AEB after 100 round broadcasting, when node number is 100, different areas

transmissions via local network topology, and does not consider to balance traffic to dissipate energy of nodes evenly. Therefore NBA (λ=0) degrades to non energy aware algorithm just same as MPR and EMPR.

AEB of NBA (λ≠0) is far less than that of MPR, EMPR and NBA (λ=0). Moreover, with the increasing of λ, AEB of NBA (λ≠0) decreases correspondingly. AEB of NBA (λ=5) is the least in the experiment. Furthermore, ARE of NBA (λ≠0) is far more than that of MPR and EMPR, and little less than that of NBA (λ=0), that is the cost paying for balancing traffic.

NFR of NBA (λ=0) is far less than that of MPR and EMPR, and NFR of NBA (λ≠0) is zero.

MRE of MPR, EMPR and NBA (λ=0) is zero. MRE of NBA (λ≠0) is far more than that of the other algorithms. Moreover, with the increasing of λ, MRE of NBA (λ≠0) increases correspondingly.

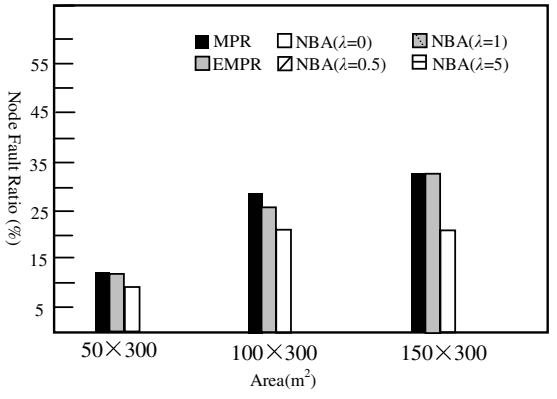


Fig. 3. NFR after 100 round broadcasting, when node number is 100, different areas

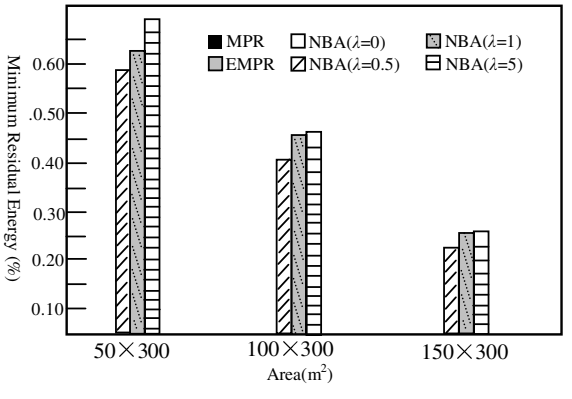


Fig. 4. MRE after 100 round broadcasting, when node number is 100, different areas

From above results, we can conclude as follows:

1) ARE of NBA is far more than that of MPR and EMPR.

The reason is that: NBA constructs a stable infrastructure and assigns type for each node via clustering algorithm at first, and then NBA decides whether nodes to do transmissions according to the properties of nodes. As nodes in WSN almost being static, little cost is paid for constructing the infrastructure. Moreover, NBA makes use of the position of node to reduce redundant MM nodes greatly when appointing MM nodes to do transmissions. As position is important for emergency and easy to get for nodes which either are equipped with GPS or get positions via location algorithms in WSN applications, it is convenient for NBA to acquire the positions of nodes.

2) ARE of NBA ($\lambda \neq 0$) is far more than that of MPR and EMPR, and AER of NBA ($\lambda \neq 0$) is far less than that of MPR and EMPR. NFR of NBA ($\lambda \neq 0$) is zero, and MRE of NBA ($\lambda \neq 0$) is far more than that of MPR and EMPR.

The reason is that: when clustering and deciding transmissions, NBA considers not only the local topology but also the energy status of nodes. Therefore, NBA ($\lambda \neq 0$)

reduces the transmissions and balances traffic, so energy is conserved and dissipated evenly throughout WSN. Thus fault nodes are deceased, and the lifetime of WSN is prolonged greatly.

In addition, we also can see that the message complexity of each node in MPR and EMPR is $O(2)$ and in NBA is $O(6)$. The reason is that NBA pays more cost to measure residual energy and balance traffic. However, how to use energy efficiently is always the most important factor for WSN to consider, so the cost is worthy.

6 Conclusion

In this paper, we propose a clustering based energy aware broadcasting algorithm, called as NBA, which reduces transmissions and balances traffic in WSN greatly. Making use of characteristics of WSN efficiently, NBA considers not only the local network topology to reduce redundant transmissions but also the energy status of nodes to dissipate energy evenly in WSN. Simulations show that NBA is much efficient and energy is conserved and fault nodes are deceased heavily, so the lifetime of WSN is prolonged greatly. And furthermore, NBA outperforms MPR and EMPR greatly in WSN.

References

1. Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: A Survey on Sensor Networks. IEEE Communications Magazine, Vol. 40, No. 8, (2002) 102–114.
2. Niculescu, D., Americ, N.: Communication Paradigms for Sensor Networks. IEEE Communications Magazine, Vol. 43, No. 3, (2005) 116–122.
3. Du, K., Wu, J., Zhou, D.: Chain-Based Protocols for Data Broadcasting and Gathering in the Sensor Networks. Proceedings of the International Parallel and Distributed Processing Symposium (2003) 1–8.
4. Ni, S. Y., Tseng, Y. C., Chen, Y., Sheu J.: The Broadcast Storm Problem in a Mobile Ad Hoc Network. Proceedings of the Fifth Asmual ACM/ IEEE International Conference on Mobile Computing and Network (1999) 151–162.
5. Tang, Y., Zhou, M., Zhang, X.: Overview of Routing Protocols in Wireless Sensor Networks. Journal of Software,Vol. 17, No. 3, (2006) 410–421.
6. Lim, H., Kim, C.: Flooding in Wireless Ad Hoc Networks. Computer Communications, Vol. 24, No. 3̃4, (2001) 353–363.
7. Williams, B., Camp, T.: Comparison of Broadcasting Techniques for Mobile Ad Hoc Networks. Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing (2002) 194–205.
8. Tian, D., Georganas, N.: A Coverage-preserving Node Scheduling Scheme for Large Wireless Sensor Networks. Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applications (2002) 32–41.
9. Heinzelman, W., Chandrakasan, A., Balakrishnan, H.: An Application-specific Protocol Architecture for Wireless Microsensor Networks. IEEE Transaction on Wireless Communications, Vol. 1, No. 4, (2002) 660–670.
10. Clausen, T., Jacquet, P.: Optimized Link State Routing Protocol (OLSR). RFC 3626 (2003).
11. Wu, J.: An Enhanced Approach to Determine a Small Forward Node Set Based on Multipoint relay. Proceedings of IEEE Vehicular Technology Conference (2003) 2774–2777.

Middleware Services for Pervasive Grids

Mario Ciampi, Antonio Coronato, and Giuseppe De Pietro

ICAR-CNR, Via P. Castellino 111, 80131 Napoli, Italy
{mario.ciampi, antonio.coronato,
giuseppe.depietro@na.icar.cnr.it}

Abstract. Grid computing and pervasive computing have rapidly emerged and affirmed respectively as the paradigm for high performance computing and the paradigm for user-friendly computing. These two worlds, however, can no longer be separated islands. As a matter of fact, pervasive and grid computing communities can both benefit from joining the two paradigms. This conjunction is already taking place. This paper presents a middleware for pervasive grid applications. It consists of a set of basic services that aim to enhance classic grid environments with mechanisms for: i) integrating mobile devices in a pervasive way; ii) providing context-awareness; and, iii) handling mobile users' sessions. Such services are OGSA compliant and have been deployed as grid services in order to be easily integrated in classic grid environments.

1 Introduction

During the last decade, new computing models have emerged and rapidly affirmed. In particular, terms like Grid Computing and Pervasive Computing have become of common use not only in the scientific and academic world, but also in business fields.

The Grid computing model has demonstrated to be an effective way to face very complex problems. The term "The Grid" was primarily introduced by Foster and Kesselman to indicate a distributed computing infrastructure for advanced science and engineering [1]. Successively, it has been extended to denote the virtualization of distributed computing and data resources such as processing, network bandwidth and storage capacity to create a single system image, granting users and applications seamless access to vast IT capabilities. As a result, Grids are geographically distributed environments, equipped with shared heterogeneous services and resources accessible by users and applications to solve complex computational problems and to access to big storage spaces.

Differently, the goal for Pervasive Computing is the development of environments where highly heterogeneous hardware and software components can seamlessly and spontaneously interoperate, in order to provide a variety of services to users independently of the specific characteristics of the environment and of the client devices [2]. Therefore, mobile devices should come into the environment in a natural way, as their owner moves, and transparently, that is owner will not have to carry out manual configuration operations for being able to approach the services and the resources, and the environment has to be able to self-adapt and self-configure in order to host incoming mobile devices.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 485 – 498, 2006.
© Springer-Verlag Berlin Heidelberg 2006

These two worlds are now evolving towards a common paradigm, namely the Pervasive Grid Computing [8]. As a matter of fact, from the Grid Computing community point of view, it's now time to integrate mobile devices into the grid because they are becoming of common use for accessing to services in any distributed environment. Moreover, it is possible to enhance the QoS of existing Grid services with characteristics, like the context-awareness and the pro-activity, which are proper of the Pervasive Computing. On the other hand, Pervasive Computing environments can proficiently benefit from grid technologies both to interconnect existing and emerging pervasive environments and to build and deploy new services that require high performance computing and large data resources [7].

The rest of the paper is organized as follows. Section 2 discusses some motivations and related work. Section 3 describes the architecture of the middleware infrastructure. Section 4 details the basic services. In section 5, a running scenario is presented. Finally, section 6 concludes the paper.

2 Motivations and Related Work

2.1 Motivations

Mobile and wireless devices have not been considered, for a long, as useful resources by traditional Grid environments. However, considering the Metcalfe's law, which claims that usefulness of a network-based system proportionally grows with the number of active nodes, and also considering that mobile devices capabilities have substantially be improved over the time, it can justifiably be stated that mobile and wireless devices are now of interest for the Grid community [3].

In particular, they have to be incorporated into the Grid either as service/resource consumers or as service/resource providers [5].

However, integration is not costless [6]. This is mainly due to the consideration that current Grid middleware infrastructures don't support mobile devices, not only because they have not been devised taking into account pervasive requirements like spontaneity, transparency, context-awareness, pro-activity, and so on, but also for three main reasons: i) they are still too much heavy with respect to mobile and wearable equipments; ii) they are not network-centric; i.e. they assume fixed TCP/IP connections and do not deal with wireless networks and other mobile technologies; and, iii) they typically support only one interaction paradigm, that is SOAP messaging, whereas the Pervasive model requires a variety of mechanisms [9].

2.2 Other Related Work

Over the last years, some valuable efforts have been done in order to make Grid architectures able to support wireless technologies and mobile devices. In particular, the paradigm of Mobile Grid or Wireless Grid has been proposed [3-6]. More recently, this paradigm has evolved in the Pervasive Grid model [7-8], which again aims at making Grid environments able to integrate mobile devices, but in a pervasive way, that is seamlessly and transparently. In addition to this, the final objective is to enhance Grid environments with characteristics like context-awareness and pro-activity that are typically found in Pervasive environments.

This effort has officially been formalized in 2003 when a Global Grid Forum Research Group, called Ubicomp-RG, was established in order to explore the possibilities of synergy between Pervasive and Grid communities.

Some interesting work towards the realization of Pervasive Grids has been done and here is reported.

In [4] mobile devices are considered as active resources for the Grid. In particular, authors developed a software infrastructure for deploying Grid services on mobile nodes and making them active actors in the Grid. This solution relies on a lightweight version of the .NET framework, namely the .NET Compact Framework, which enables to deploy on mobile devices simple Grid Services that require limited amount of resources. It is important to note that such a solution applies only to mobile devices equipped with the Microsoft Pocket PC operating system and requires several manual operations for installation and configuration.

In [9] authors argue that the SOAP messaging, which is the current interaction paradigm for standard grid middleware infrastructures, is not adequate to face needs of pervasive grids. They developed several plug-ins and an handling component for enlarging the set of available interaction mechanisms in order to make grids able to support heterogeneous software components.

Another middleware infrastructure for pervasive grids have been presented in [10]. In that case, authors have concentrated their effort on the extension of existing resource manager components in grid applications for making them able to register mobile devices.

2.3 Our Contribution

Our contribution consists of a set of basic services that realize a middleware infrastructure, namely MiPeG, for the realization of Pervasive Grids. In particular, MiPeG extends existing Grid environments with the following characteristics:

a. **Spontaneous and transparent integration of mobile devices** – This characteristic makes the grid able to integrate mobile devices in a pervasive way. In particular, integration takes place without any manual configuration operation. The realized services grant access to mobile devices. In addition to this, they reliably handle implicit disconnections of mobile devices. Indeed, a mobile user can leave the environment without concerning about pending services. In this case, the environment detects disconnections and handles pending computations.
b. **Context-awareness** – Some mechanisms for context-awareness and location-awareness are provided. In particular, we developed services for i) locating mobile users in a physical area; ii) tracking their activities within the environment; and, iii) personalizing their access based on rights, device capabilities, location, and so on.
c. **Management of mobile user's sessions** – As users physically move in the environment, by carrying their devices, their running applications, as well as their virtual environments, are updated and made available for them in the new location. This is performed by services that handle mobile sessions.

3 Architecture of MiPeG

The middleware that we have developed consists of a set of basic services, as shown in figure 1. Such services are exposed as Grid Services; i.e., they are compliant with the OGSA specifications [11].

MiPeG integrates with the Globus Toolkit [12], which is the the-facto standard platform for Grid applications, in order to extend its functionalities and to provide mechanisms for augmenting classic grid environments with pervasive characteristics. It also uses the JADE framework [13] to implement some mobile agent based components.

It is important to note that MiPeG supports two main interaction mechanisms: the SOAP messaging, for Web and Grid Services, and a publish-subscribe model for more generic, inter-component, asynchronous communications.

MiPeG consists of the following basic services:

- *AsynchronousCommunicationBroker* – This component is in charge of dispatching asynchronous events in the pervasive grid environment. It implements the WS-BrokeredNotification specification [14]. Moreover, it extends such a specification by classifying events and handling hierarchies of classes of events. This extension provides a more flexible and efficient mechanism for subscriptions and communications.
- *Access&LocationService* – This service provides network access to mobile devices and locates them in the environment [16]. Current implementation grants access and locates 802.11 WiFi enabled devices. It also locates RFID tagged objects [17]. Moreover, we are developing and integrating mechanisms for granting access and locating Bluetooth enabled devices. Definitively, this service is in charge of communicating to the environment i) incoming mobile objects, ii) location changes for active objects, and iii) leaving objects.
- *SessionManagerService* – This service handles sessions for mobile devices. It consists of mobile agents that maintain the list of services activated by mobile users. A mobile agent is created when a new device comes in the environment, required to move accordingly with device movements, and destroyed when the mobile device leaves the environment.
- *DeviceService* – This service handles the list of mobile resources active in the environment.
- *PeopleService* – This service provides basic authentication mechanisms and handles the list of mobile users active in the environment.

All software elements are full open-source Java components. Moreover, most of them have a WSDL interface, consist in a GAR file generated with the Ant tool, and are deployed as Grid Services over the Globus Toolkit 4.0 platform. In addition, some services present Web interfaces realized with the java JSP and Servlet technologies and interact with MySQL databases.

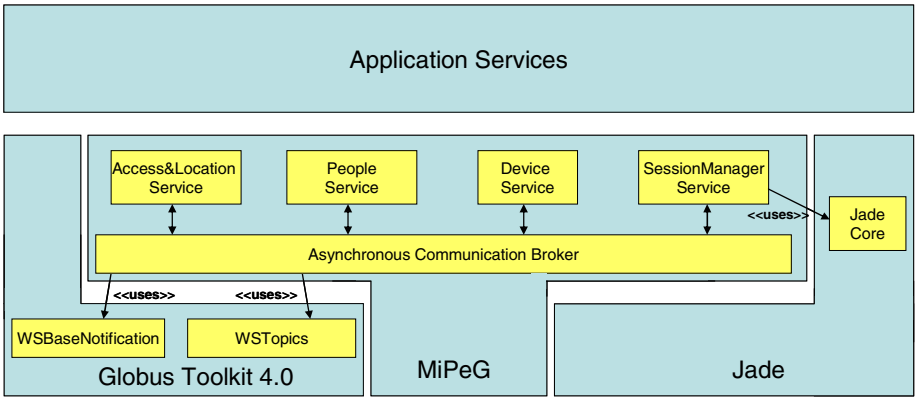


Fig. 1. Architecture of MiPeG

4 Details on MiPeG's Services

4.1 The Asynchronous Communication Broker

The Asynchronous communication service provides an interaction mechanism based on the event publish-subscribe paradigm.

It implements the WSBrokeredNotification specifications. It also extends such specifications with the possibility of publishing and subscribing hierarchies of classes of events.

In particular, a class of events is a collection of events that are logically related. In this case, rather than subscribing every event of the class, a consumer can make a unique operation of subscription for the entire class and be notified for every event.

Moreover, it handles hierarchies of classes. The concept of hierarchy has been derived from the Object Oriented inheritance technique. Since, classes can be related in hierarchies, when a consumer subscribes one class, it automatically subscribes every upper class of the hierarchy. An example is shown in figure 2. In the figure, Consumer1 and Consumer2 respectively subscribe classes C1 and C6. Because of the inheritance property of the hierarchy, Consumer1 is notified only for events appertaining to class C1. Differently, Consumer2 is notified for events of every class except C5. This is due to the fact that class C6 inherits from classes C2, C3, and C4,

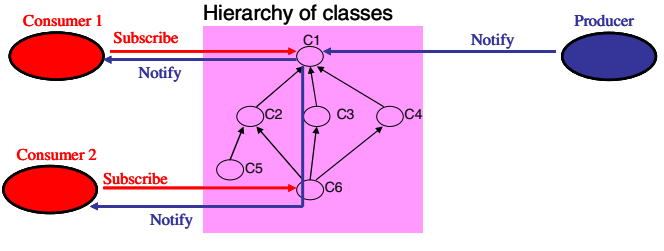


Fig. 2. A possible hierarchy of classes of events

which in turn inherit from class C1. In particular, it is shown the case of a producer that notifies an event of class C1. This event is dispatched both to Cosumer1 and to Consumer2.

This model represents an extension of the WSBrokeredNotification specifications. As a matter of fact, the WSBrokeredNotification specifications neither support hierarchies of classes nor simple classes, but require that a consumer subscribe every specific event for which it wants to be notified.

In figure 3, the WSDL interface of the Asynchronous Communication Service is shown. The Broker exposes functions for subscribing and unsubscribing classes of events, notifying events, creating and handling hierarchies. In addition, two hierarchies, which will be used in the running scenario, are detailed. In particular, hierarchy 1 consists of three classes. Class MOBILITY contains the event NEW_LOCATION. Class PRESENCE contains the events NEW_DEVICE and DEVICE_HAS_LEFT. Finally, class LOCALIZATION groups all events. Hierarchy 2 consists of a single class, namely USER_PRESENCE, which contains events for logging in and out an user.

Further information on the proposed model and the realized service can be found in [15].

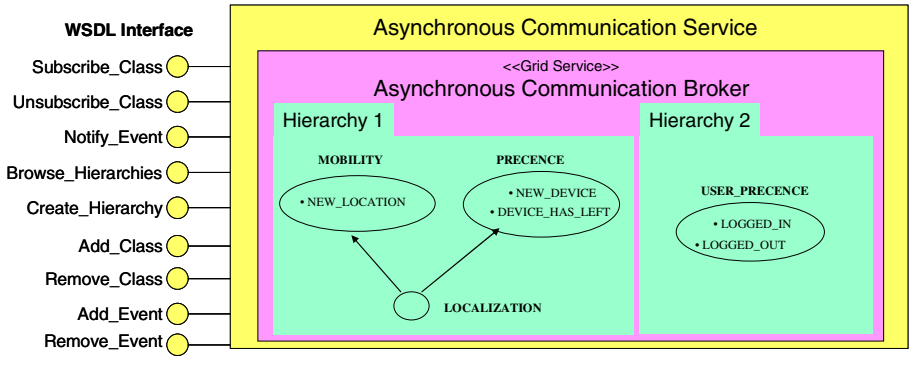


Fig. 3. Interfaces and architecture of the ACB

4.2 Access&Location Service

This service is able to locate active mobile objects like WiFi enabled devices and RFID tagged entities. It offers both locating and location functions; that is, the function *Locate_Object* returns the position of a specific object, whereas the function *Get_Objects* returns the list of objects that are actives at a specific location.

It notifies to the environment events of the class LOCALIZATION.

In addition to location and locating functions, this service provides basic network connectivity facilities for incoming mobile devices and detects leaving mobile objects.

The service architecture consists of two layers and the following components:

- *WiFiLocatingComponent* – This component is in charge of locating WiFi enabled mobile devices. In particular, a WiFi location is identified by the area

covered by a specific wireless Access Point (AP). In the environment, one *WiFiLocatingComponent* is deployed per every wireless AP. Current implementation uses 3Com Office Connect Wireless 11g Access Points. Whenever a mobile device connects with the AP, the AP writes an event in its log file. The *WiFiLocatingComponent* periodically interrogates such a log file and communicates to the *LocationComponent* when a new device has connected. The *WiFiLocatingComponent* maintains information on devices locally connected to its AP.

- *RFIDLocatingComponent* – This component is in charge of locating RFID tagged objects. An RFID location is identified by the area covered by a specific RFID reader. Current implementation uses the passive, short-range (30 cm), Feig Electronic RFID, model ISC.MR 100/101. When a tagged object enters the area covered by an antenna, the RFID reader generates an event that is caught by the *RFIDLocatingComponent*. Then, the *RFIDLocating-Component* communicates to the LocationComponent such an event.
- *DHCPComponent* – This component implements a DHCP service. It provides network connectivity to incoming IP enabled devices as a standard DHCP, but it has also additional functionalities. In particular, it is able to release an IP address on demand. By this way, if a device has left the environment, the *LocationComponent* requires that the *DHCPComponent* free the IP address of that device.
- *EcoComponent* – This component sends ping messages towards mobile IP devices in order to detect leaving objects. When an implicit disconnection is detected, the component communicates such an event to the *LocationComponent* that notifies the DEVICE_HAS_LEFT event.
- *LocationComponent* – This component is in charge of handling global location states obtained by combining information coming from Locating components. When a mobile object changes its position the NEW_LOCATION event is notified. Moreover, when a mobile object is detected for the first time by a LocatingComponent, it is added to the active device list and the NEW_DEVICE event is notified.

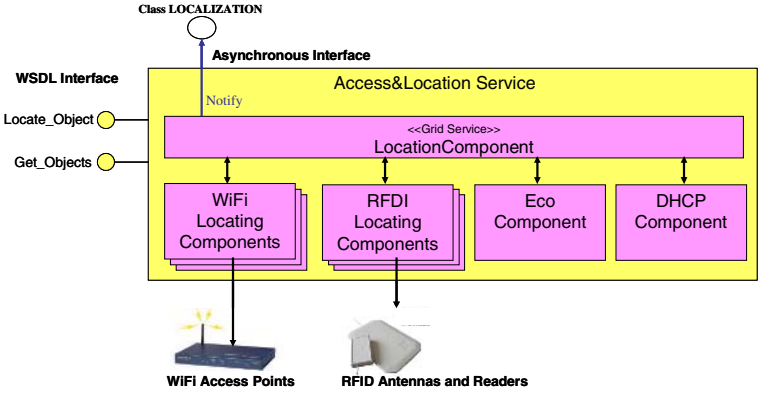


Fig. 4. Interfaces and architecture of the Access&Location Service

4.3 People Service

This service provides basic authentication mechanisms and handles active users in the environment. As shown in figure 5, it exposes a Web interface for authenticating connected users and for handling authorized users. An authorized user can login by means of a jsp form application and logout either explicitly or implicitly by closing the web page. moreover, some administrative functions are available. The environment can interact with the service via the WSDL interface in order to get the list of active users or the rights of a specific user.

The service also notified to the environment whenever a user logs in or logs out.

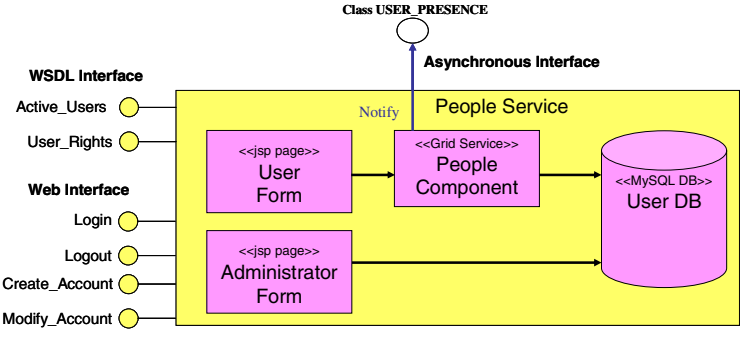


Fig. 5. Interfaces and architecture of the People Service

4.4 Device Service

This service handles a list of active mobile resources. In particular, when a new mobile device is detected it is inserted in the list of active mobile resources. The mobile device is removed from the list when it is no longer available in the environment.

The service subscribes the class LOCALIZATION in order to be notified for new devices and for devices that have changed their position or have left the environment.

Finally, the Device Service also exposes WSDL functionalities for looking up in the list of active devices.

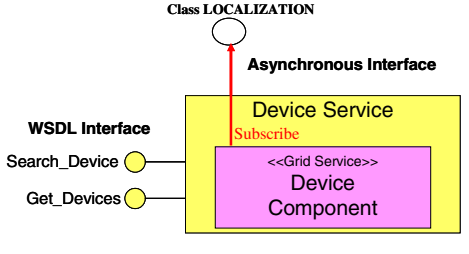


Fig. 6. Interfaces and architecture of the Device Service

4.5 Session Manager Service

The *SessionManagerService* has two major functions. It handles both the list of services available at every location of the environment (in a pervasive scenario some services could not be available everywhere but only at specific locations) and the list of services activated by every user.

As shown in figure 7, it consists of the following components:

- *LocationAgent* – This is an agent deployed at a specific location. There's a *LocationAgent* per every location. It handles the list of application services available at its location. Services availability may depend not only on the physical location but also on user's rights and profile. In particular, several levels of access are defined.
- *PersonalAgent* – This is a mobile agent that is created when a new device appears in the environment. In particular, the *SessionManagerComponent*, which is notified for the NEW_DEVICE event, creates the *PersonalAgent* at the location where the mobile device has been detected. After its creation, the *PersonalAgent* subscribes the class USER_PRESENCE and interacts with the local *LocationAgent* to have the list of available services. From this moment on, the *PersonalAgent* catches user's requests and maintains the list of services activated by him. In the case the user authenticates himself, the *PersonalAgent* is notified by the LOGGED_IN event, then interacts once again with the *LocationAgent* to have the new list of available services. When a device moves in a new location, the *SessionManagerComponent*, which receives the NEW_LOCATION event, requires that the *PersonalAgent* migrate and update the list of services available at the new location. Finally, in case the user logs out or his device disconnects, the PersonalAgent handles pending computations.
- *SessionManagerComponent* – This component is an agent container. It creates both Personal and Location Agents. It also drives Personal Agents to move accordingly with owner movements. In particular, it creates a new *PersonalAgent* when it receives a NEW_DEVICE event; it requires that the *PersonalAgent* move to a new location when it catches a NEW_LOCATION event; finally, it requires that the *PersonalAgent* destroy itself when the DEVICE_HAS_LEFT event is notified.

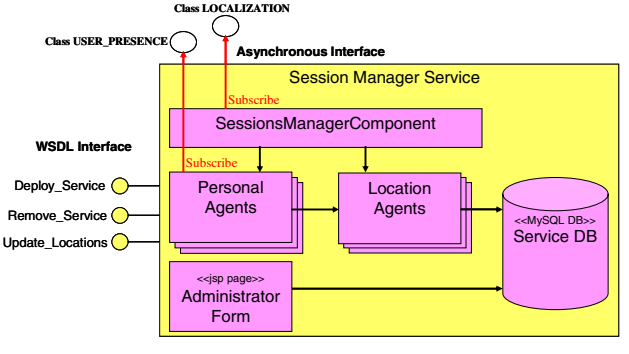


Fig. 7. Interfaces and architecture of the Session Manager Service

All components for the *SessionManagerService* have been developed over the JADE framework, which is fully compliant with the FIPA specifications [18].

5 Experimental Scenario

The experimental scenario consists of a physical site located in a three floors building. The virtual environment uses two floors of the building.

Floor zero has a computing laboratory in which a cluster of 24 linux PCs, a 12 processors Silicon Graphics workstation, and a motion capture system are deployed. Such resources are collected in a wired grid built at the top of the Globus Toolkit 4.0 platform.

On floor two, wireless access to the grid is available. As a matter of fact, two 3Com Office Connect Wireless 11g Access Points identify two distinct locations. L1 is a student laboratory where our students develop their activities and periodically perform E-Tests. L2 is a multimedia room equipped with a projector, an interactive monitor, and other multimedia devices.

Some services are available:

- *MotionCaptureService* – This service relies on the motion capture system. An actor (equipped with optical markers) moves around in the multimedia laboratory. Several cameras capture his movements that are reproduced on a graphic station. The graphic station shows a skeleton, which moves accordingly with the actor, and records data movement in a file;
- *RenderingService* – This service enables users to submit row motion data and to build 3D graphic applications. This service is exposed as a Grid Service and is available at every location (L1, L2);
- *PresentationService* – This service enables a user to project its presentation in the multimedia room. The service receives a pdf/ppt file via a dialog form and then enables speaker to control the presentation flow. This is an interactive service, which requires the speaker to be in the room for presentation. As a consequence, the service must be available only in the multimedia room (L2);
- *ETestingService* – This service performs on-line evaluation tests for courseware activities. When a session test starts, students must be in the student laboratory. Evaluation tests are synchronized and students have a predefined period of time for completing each test section. Students can interrupt their test by explicitly closing the service or by leaving the multimedia room. This service is exposed as a Grid Service, but it must be available only in the student laboratory (L1);

In this environment, services availability depends on user location, rights, and context. As a matter of fact, we can report some example scenarios:

1. The *PresentationService* is available only to users that are located in L2. In particular, a mobile user, who moves in location L2, is followed by his *PersonalAgent*. The *PersonalAgent* interacts with the *LocationAgent* and updates the list of available services. From now on, the mobile user can get access to the *PresentationService*.

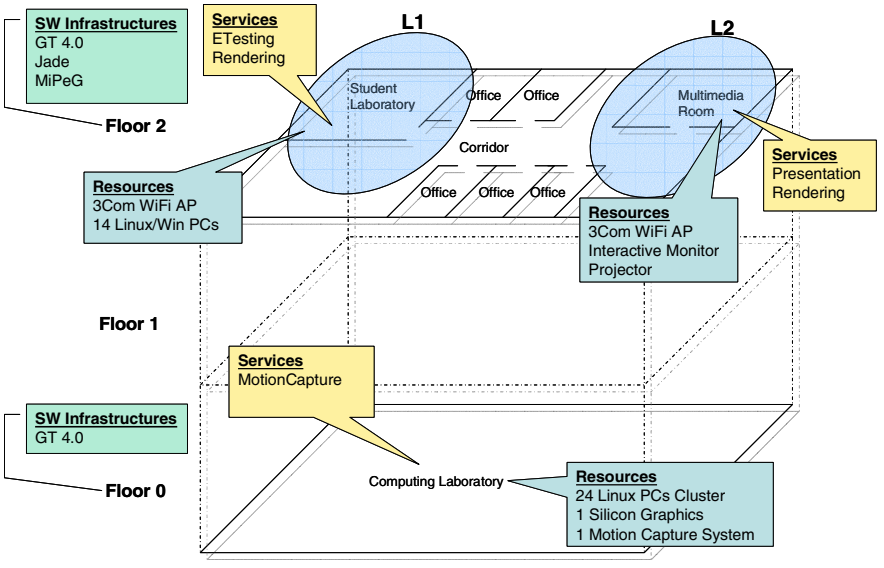


Fig. 8. Real environment

2. The *ETestingService* is available for every authenticated mobile user within the student laboratory. In particular, while a test session is active, a mobile user that enters in location L1 can get access to the service (after having authenticated himself). On the other hand, if he leaves the location L1, the *ETestingService*, which is notified by the *Access&LocationService*, automatically disconnects the leaving user and denies any further tentative of reconnection.
3. The *RenderingService* is available for authorized users at every location. However, the QoS is improved by context-awareness. Indeed, in the case a mobile user launches such a service while in location L1, rendered data are reproduced on his mobile device. After that, if the user moves in the multimedia room (L2), rendered data are automatically switched on the interactive monitor (if idle).

We have monitored the access to application services for three weeks. As reported in table 1, we have registered 155 total accesses and 49 accesses through mobile

Table 1. Number of accesses to application services

Service	Mobile Accesses	Wired Accesses	Total Accesses
Motion Capture Service	0	4	4
Rendering Service	7	20	27
Presentation Service	6	0	6
E-Testing Service	36	82	118
Total	49	106	155
Percentage	31,61%	68,39%	

devices (32%). Figure 9 represents such results. It is worth noting that access through mobile devices is granted because the application services are equipped with multi-channel interfaces and because MiPeG's basic services enable mobile devices to connect in a transparent and spontaneous way and the environment to efficiently handle and locate them. Therefore, a classic grid infrastructure wouldn't be able to offer services to mobile devices.

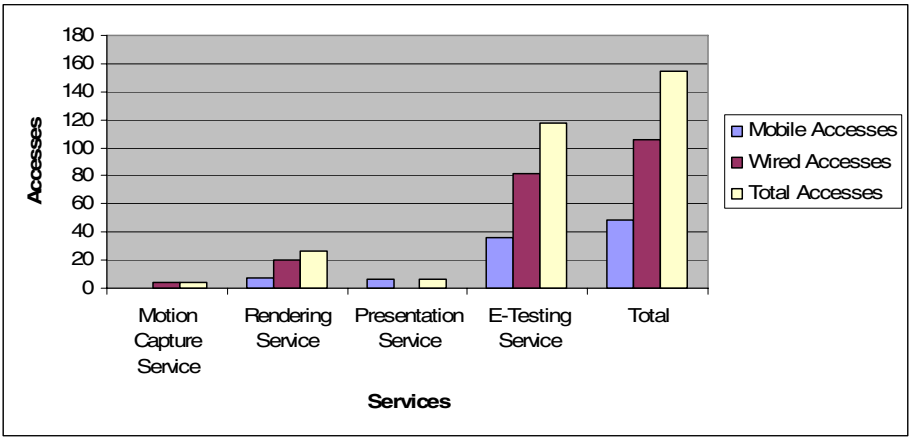


Fig. 9. Distribution of accesses

In the previous examples, we have shown how our Grid environment already benefits of developed pervasive characteristics.

In addition, we are realizing an advanced multimedia service, for immersive virtual worlds, by integrating 1) the *MotionCaptureService*, 2) the Real-Time *RenderingService*, 3) RFID locating technologies, and other wearable equipments like multimedia helmets and gloves. In particular, the service consists of a virtual world, which is visualized by a mobile user through an helmet. The mobile user interacts with the virtual world, which is rendered in real time by a Grid service. User actions and movements are caught by the RFID locating mechanism and notified to the rendering service, which consequently builds a new scene. This is just an example of new application service that can be realized by combining Pervasive and Grid Computing.

6 Conclusions

This paper has described an ongoing middleware for pervasive grids. The middleware provides basic services for integrating mobile devices in the grid and for extending the grid with context-awareness.

Currently, integration of mobile devices has taken place mostly for enabling mobile users to get access to grid services in a pervasive way. Future work will aim at integrating mobile devices as service providers. In this case, mobile devices would act not only as interfaces towards the grid, but they would be perceived as available

resources for the grid itself, that is they could receive code to be executed or host special services (say for an example measurement systems) for the grid. Obviously, this requires the development of advanced mechanisms for resource management that take into account issues related to the mobility of such devices like intermittent network connections, battery dependency, and so on. In order to achieve this target, we are 1) extending the Glue Schema that is the standard information model for the grid's resources and making it able to take into account properties of mobile devices not considered until now; 2) developing monitoring tools (like the Gram) to catch the mobile devices state; and 3) extending the GT4's Monitoring&Discovery Service to actually integrate mobile devices into the grid.

References

[1] Foster, C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure". Morgan Kaufmann, 1999.

[2] D. Saha and A. Murkrjee, "Pervasive Computing: A Paradigm for the 21st Century", IEEE Computer, March 2003.

[3] L. W. McKnight, J. Howinson, S. Bradner, "Wireless Grids", IEEE Internet Computing, July-August 2004.

[4] D. C. Chu and M. Humphrey, "Mobile OGSI.NET: Grid Computing on Mobile Devices", International Workshop on Grid Computing, GRID 2004.

[5] B. Clarke and M. Humphrey, "Beyond the 'Device as Portal': Meeting the Requirements of Wireless and Mobile Devices in the Legion of Grid Computing System", International Parallel and Distributed Processing Symposium, IPDPS 2002.

[6] T. Phan, L. Huang and C. Dulan, "Challenge: Integrating Mobile Devices Into the Computational Grid", International Conference on Mobile Computing and Networking, MobiCom 2002.

[7] N. Daves, A. Friday, and O. Storz, "Exploring the Grid's Potential for Ubiquitous Computing", IEEE Pervasive Computing, April-June 2004.

[8] V. Hingne, A. Joshi, T. Finin, H. Kargupta, E. Houstis, "Towards a Pervasive Grid", International Parallel and Distributed Processing Symposium, IPDPS 2003.

[9] G. Coulson, P. Grace and G. Blair, D. Duce, C. Cooper and M. Sagar, "A Middleware Approach for Pervasive Grid Environments", Workshop on Ubiquitous Computing and e-ResearchNational eScience Centre, Edinburgh, UK 18-19 May 2005.

[10] C. F. R. Geyer et. al., "GRAPEp: Towards Pervasive Grid Executions", III Workshop on Computational Grids and Applications, WCGA 2005.

[11] http://www.globus.org

[12] Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems", IFIP International Conference on Network and Parallel Computing, Springer-Verlag LNCS 3779, pp 2-13, 2005, also available on-line at: www.globus.org.

[13] F. bellifemmine, A. Poggi and G. Rimassa, "Jade Programmers Guide", http://sharon.cselt.it/projects/jade

[14] http://www.oasis-open.org/committees/download.php/13485/wsn-ws-brokered_notification-1.3-spec-pr-01.pdf

[15] M. Ciampi, A. Coronato, G. De Pietro, and M. Esposito, "Handling Structured Classes of Events in Pervasive Grids", to appear in the proc. of the 4th International Metainformatics Symposium, MIS 2005, as Lecture Notes in Computer Science, LNCS, Springer Verlag.

[16] Coronato, M. Ciampi, G. De Pietro, and M. Esposito, "A Location Service for Pervasive Grids", in the proc. of the International Conference on Systems, Computing Sciences and Software Engineering (SCS2 2005).

[17] Coronato, G. Della Vecchia, and G. De Pietro, "An RFID-Based Access&Location Service for Pervasive Grids", in the proc. of the 1st International Workshop on Thrustworthiness, Reliability and services in Ubiquitous and Sensor neTworks (TRUST 2006) as Lecture Notes in Computer Science, LNCS 4097, Springer Verlag.
[18] http://www.fipa.org/specifications/index.html.

Allocating QoS-Constrained Workflow-Based Jobs in a Multi-cluster Grid Through Queueing Theory Approach

Yash Patel and John Darlington

London e-Science Centre, Imperial College, London
{yp03, jd}@doc.ic.ac.uk

Abstract. Clusters are increasingly interconnected to form multi-cluster systems, which are becoming popular for scientific computation. End-users often submit their applications in the form of workflows with certain Quality of Service (QoS) requirements imposed on the workflows. These workflows describe the execution of a complex application built from individual application components, which form the workflow tasks. This paper addresses workload allocation techniques for Grid workflows. We model individual clusters as $M/M/k$ queues and obtain a numerical solution for missed deadlines (failures) of tasks of Grid workflows. The approach is evaluated through an experimental simulation and the results confirm that the proposed workload allocation strategy combined with traditional scheduling algorithms performs considerably better in terms of satisfying QoS requirements of Grid workflows than scheduling algorithms that don't employ such workload allocation techniques.

1 Introduction

Clusters are becoming important contenders for both scientific and commercial applications. Clusters with different performance and architectures, owned by different organisations are now increasingly interconnected to form a multi-cluster computing system [12].

Complex scientific experiments within a Grid are increasingly specified in the form of workflows, which detail the composition of distributed resources such as computational devices, data, applications, and scientific instruments [3]. Users who submit a workflow to the Grid will often have constraints on how they wish the workflow to perform. These may be described in the form of a Quality of Service (QoS) document which details the level of service they require from the Grid. This may include requirements on such things as the overall execution time for their workflow; the time at which certain parts of the workflow must be completed; cost to the user. In order to determine if these QoS constraints can be satisfied it is necessary to store performance information of resources and applications within the Grid. Such information could also be performance data for software to be run on a computational resource; resource information about speed and reliability, mean service time and mean arrival rate. Here we see that existing Grid middleware for resource descriptions [20] and performance repositories [6] may be used for the storage and retrieval of this data.

Scheduling within Grid is mainly based on two techniques. Either scheduling is performed based on real time information such as waiting time in the queue, residual processing time; or on average-based metrics such as mean service rate, mean arrival rate.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 499–510, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Real time information based algorithms generally perform better than average-based strategies [21]. However, obtaining real time information from a distributed system such as Grid, leads to high overheads. Moreover resources may be distributed geographically, which means that obtaining instantaneous information about the states of geographically distributed resources can lead to substantial delays and consequently to inaccurate scheduling decisions. Also, it may not be possible to obtain instantaneous information at any arbitrary point in time for some distributed systems. Thus, it is necessary to develop approaches which are not dependent on obtaining accurate instantaneous information. The use of average-based strategies seems to be an appropriate approach. Average-based scheduling, for jobs based on FCFS (First Come First Served) rule in a Grid, consists of two stages. The first stage deals with distributing the workload received by a central entity such as a brokering service, which connects several clusters. This process is referred to as workload allocation strategy in this paper. The second stage deals with dispatching jobs to resources underneath the clusters using appropriate scheduling algorithms [15]. This second stage is referred to as job dispatching strategy in this paper. The workload allocation scheme determines the proportion of workload directed to each cluster, while the job dispatching strategy distributes the incoming jobs to cluster resources in a way that satisfies the QoS requirements of the jobs. This paper is organised as follows. Section 2 presents related work and compares our work with others in the field. Section 3 describes the Grid model (figure 1) considered and assumptions held in this paper. Workload allocation strategy in terms of minimising workflow task failures is obtained numerically in Section 4 and the performance of the workload allocation and scheduling strategy is evaluated in Section 5. Finally, we conclude the paper in Section 6.

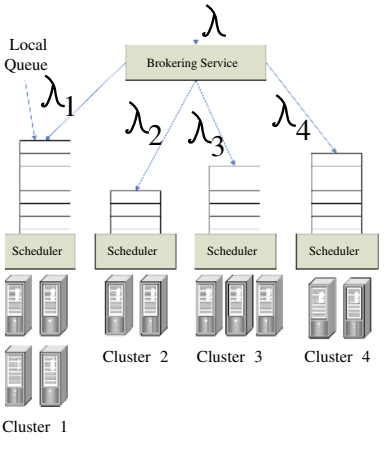


Fig. 1. Grid Model

2 Related Work

Multi-cluster systems have been studied widely and there is considerable research material available [8] [16] [12]. Projects such as EGEE (Enabling Grids for E-Science)

[2] aim to integrate various national, regional and thematic Grid efforts, in order to create a seamless multi-cluster Grid infrastructure for the support of scientific research. It has been shown that it is hard to obtain an optimal workload allocation in a distributed and heterogeneous system such as Grid [21] [17]. Banawan et al. [18] develop an optimization function for allocating workload to resources. However, the solution to the objective function is not given and the optimization function is limited to a single cluster only and does not extend to distributed systems such as Grid. A workload allocation technique is developed by Tang et al. [21], which aims to optimize response times in a heterogeneous cluster. Tang et al. obtain both an objective function and its solution. However they assume that each cluster has only one computing node.

Using clusters to process jobs with QoS requirements is becoming propular as scientific experiments and projects increasingly make use of distributed systems such as Grid to satisfy their compute-intensive needs [7] [9]. Kao et al. [4] use two homogenous non real time servers to provide a service that will satisfy QoS requirements of jobs. Zhu et al. extend the work of Kao et al. by using a homogeneous cluster that aim to satisfy QoS requirements of jobs [19]. The performance of scheduling based on minimising failures to meet waiting time requirements (the maximum time a job can wait before execution) of jobs is also evaluated in [19]. However, their work is confined to a single cluster only and does not consider a distributed system such as Grid.

Our work focuses on developing a workload allocation strategy which minimises failures of QoS-constrained workflows in a Grid. The Grid is modelled as a cluster of clusters (figure 1), where jobs in the form of workflows arrive with certain QoS requirements imposed on workflows. Relevant Grid components such as performance repository, workflow management system and others are not shown in figure 1, in order to focus on the central topic of the paper.

3 The Model

We model each cluster as an $M/M/k$ queue [10], where k is the number of computational nodes in the cluster. The mean service rate of a node in cluster i is u_i and the arrival rate is λ_i. Each cluster has its own individual scheduler. Jobs to this scheduler are dispatched via a brokering service, which in turn connects different clusters. The brokering service receives jobs in the form of workflows, which are composed of individual tasks. When a workflow task finishes, further tasks must be started. Hence the brokering service dispatches jobs of new and old workflows to appropriate clusters using a workload allocation strategy developed in the next section. The jobs are executed in the order they are received. The scheduler of a cluster sends jobs to its processing nodes using a scheduling strategy. For simplicity we have considered round-robin and random job dispatching only in our experimental evaluation. However this is not a restriction and any other traditional scheduling strategy could be used in place. The jobs received by the clusters follow a uniform distribution of deadline requirements, as assumed in [4] [19]. Workflows have overall deadline constraints, which are explicitly specified by the end-user. There could also be other constraints such as cost, reliability, network constraints. A full list of constraints is beyond the scope of this paper. However for simplicity, we keep the QoS requirements of workflows limited to overall deadline

constraints. Deadlines for individual tasks of workflows are calculated by the brokering service using a formula given in the experimental evaluation. We define a workflow failure as failure in meeting the overall workflow deadline. Failure in meeting an intermediate workflow task deadline is not a workflow failure. We thus minimise workflow failures by minimising failures of intermediate workflow tasks.

4 Theoretical Analysis of Minimisation of Workflow Task Failures

In this section we obtain a numerical solution for failure of tasks of Grid workflows in a cluster and a non-linear program for computing workload allocation for clusters. The solution of the non-linear program is the workload proportion for each cluster.

4.1 Workload Allocation Based on Failure Minimisation of Workflow Tasks

In this section, a workload allocation strategy using failure of tasks of Grid workflows for a cluster is developed. The Grid consists of n clusters, where each cluster is modelled as an $M/M/k$ queue with infinite customer capacity, meaning the number of jobs that can wait in the queue of a cluster are infinite. Hence essentially a cluster is indeed an $M/M/k/\infty$ queue. Cluster i has k_i processing nodes, where each processing node has a mean service rate of μ_i. We consider that the brokering service receives jobs with an arrival rate of λ, out of which λ_i is allocated to i^{th} cluster. Thus the arrival rate can be expressed as the sum of workload proportions of clusters, given by equation 1.

$$\lambda = \sum_{i=1}^{n} \lambda_i \tag{1}$$

Let $r_i(t)$ and $R_i(t)$ be the PDF (Probability Density Function) and CDF (Cumulative Density Function) of the response time of cluster i respectively for an $M/M/k$ queue. Response time of a job consist of its waiting time in the queue of a cluster and its service time. The PDF of the response time for an $M/M/k$ queue is given by equation 2 [10]. By definition, CDF of the response time can be represented by equations 4 and 5.

$$r_i(t) = (1 - \frac{k_i \rho^{k_i}}{k_i!(k_i - \rho)(\sum_{j=0}^{k_i-1} \frac{(k_i\rho)^j}{j!} + \frac{(k_i\rho)^{k_i}}{k_i!(1-\rho)})})e^{-\mu_i t}$$

$$-\frac{\mu_i \rho^{k_i}(e^{-\mu_i(k_i-\rho)t} - e^{-\mu_i t})}{(\sum_{j=0}^{k_i-1} \frac{(k_i\rho)^j}{j!} + \frac{(k_i\rho)^{k_i}}{k_i!(1-\rho)})(k_i - 1)!(1 - k_i - \rho)} \tag{2}$$

$$\rho = \frac{\lambda_i}{k_i \mu_i} \tag{3}$$

$$R_i(t) = P(T \leq t) \tag{4}$$

$$R_i(t) = \int_0^t r_i(t)dt \tag{5}$$

We assume a uniform distribution of deadlines of jobs to be allocated to clusters, as assumed in [4] [19]. This is not a restriction and general distributions could be accomodated as well, while still keeping the analysis effective. Let the lower and upper bounds of deadline of jobs to be allocated to clusters be L and U respectively. Its PDF $d(t)$ is given by equation 6.

$$d(t) = \frac{1}{U - L} \tag{6}$$

Now we can compute expected failures for cluster i using the following integral, given by equation 7, where $Failures_i$ are workflow task failures for cluster i. Equation 7 is an integral that computes a continuous expectation. It computes expected failures for an arrival rate of λ_i. The term $1 - R_i(t)$ is the probability of jobs to finish in time greater than t and the term $d(t)$ is the probability of the number of jobs requiring t time to finish. Thus the product of the two terms compute a failure probability, which when multiplied with λ_i yields the expected number of failures.

$$Failures_i = \lambda_i \int_L^U d(t)(1 - R_i(t))dt \tag{7}$$

Solving the integral, we obtain expected failures for cluster i, given by equation 8.

$$\begin{aligned} Failures_i &= 1 - y_1 + y_2\mu_i(k_i - \rho) - \frac{y_2}{\mu_i} \\ &+ (\frac{1}{U - L})(\frac{y_1}{\mu_i} + \frac{k_i - \rho}{\mu_i})(e^{-\mu_i L} - e^{-\mu_i U}) \\ &- (\frac{y_2}{U - L})(e^{-\mu_i L(k_i - \rho)} - e^{-\mu_i U(k_i - \rho)}) \end{aligned} \tag{8}$$

$$y_1 = \frac{1 - (\frac{1}{\sum_{j=0}^{k_i - 1} \frac{(k_i\rho)^j}{j!} + \frac{(k_i\rho)^{k_i}}{k_i!(1-\rho)}})\frac{k_i\rho^{k_i}}{k_i!(k_i - \rho)}}{\mu_i}$$

$$y_2 = \frac{\mu_i(\frac{1}{\sum_{j=0}^{k_i - 1} \frac{(k_i\rho)^j}{j!} + \frac{(k_i\rho)^{k_i}}{k_i!(1-\rho)}})\rho^{k_i}}{(k_i - 1)!(1 - k_i - \rho)}$$

In case of arbitrary distribution of deadlines of jobs, the distribution can be expressed mathematically given by equation 9.

$$P(d_1) = p_1,\ P(d_2) = p_2, ..., \ P(d_m) = p_m \tag{9}$$

The sum of probabilities of m jobs is equal to one. The terms d_1 to d_m are the deadline allocations of jobs. We can express expected failures for cluster i given by equation 10.

$$Failures_i = \lambda_i \sum_{i=1}^{n} P(t)(1 - R_i(t)) \tag{10}$$

Thus total failures expected is the sum of expected failures for all clusters, given by equation 11.

$$Failures = \sum_{i=1}^{n} Failures_i \tag{11}$$

The objective is to minimise total failures. We can now write the minimisation problem represented by equations 12 to 14.

$$minimise[\sum_{i=1}^{n} Failures_i] \tag{12}$$

subject to

$$\sum_{i=1}^{n} \lambda_i = \lambda \tag{13}$$

$$\forall i, 0 \leq \lambda_i \leq \lambda \tag{14}$$

The above problem can be solved using appropriate non-linear optimisation techniques. We solve it as constrained Lagrange multiplier problem. We use GAMS [1] language to model the non-linear problem and use dicopt [11] as the optimisation solver by Carnegie Mellon University, to solve it.

5 Experimental Evaluation

In this section we present experimental results for the workload allocation and job dispatching techniques described in this paper.

5.1 Setup

Table 1 and 2 summarise the experimental setup. We have performed 3 simulations, the first with workflow type 1, second with workflow type 2 and in the third simulation, workload is made heterogenous. The workflows experimented with are shown in figure 2. Workflow type 1 is quite simple compared to type 2, which is a real scientific workflow. In the first two simulations, the workflows are all similar but having different overall workflow deadlines. In the third simulation, workload is made heterogenous (HW), meaning any of the three workflows shown as heterogenous workload, in figure 2 could be submitted. Apart from that, the workflows have different overall workflow deadlines. Cluster setups for different simulations are shown in table 1. We have performed 10 runs in each different setup of a simulation and averaged out the values. Initially 500 jobs allow the system to reach steady state, the next 1000 jobs are used for calculating statistics such as mean execution time of a workflow, mean workflow failures and mean utilisation of a cluster. The last 500 jobs mark the ending period of the simulation. The simulation is developed on top of simjava 2 [5], a discrete event simulation package. The Grid size is kept small in order to get an asymptotic behaviour of workflow failures, as coefficient of variation (CV) of workflow task execution time

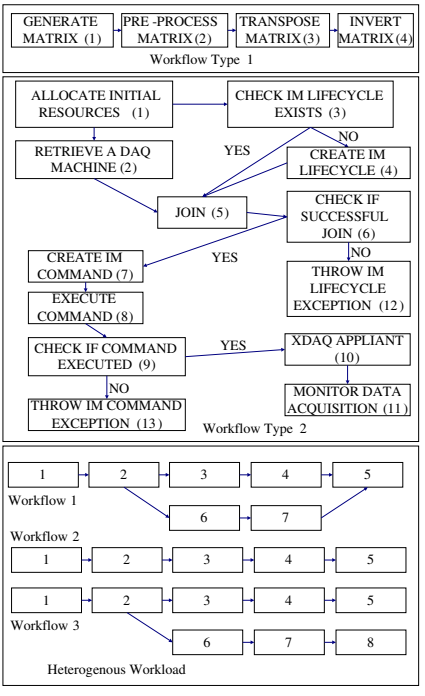


Fig. 2. Workflows

Table 1. Clusters setup

Cluster	Machines (Simulation 1/3)	Machines (Simulation 2)	Avg. Speed (MIPS)
1	6	3	14000
2	6	3	10000
3	6	3	5000
4	6	3	3000

Table 2. Simulation parameters

Simulation	1	2	3
Arrival Rate (λ) (per sec)	1.5-10	0.1-2.0	1.5-3.6
Task Mean (μ) (sec)	3-12	3-10	3-12
Task CV = σ/μ	0.2-2.0	0.2-1.4	0.2-2.0
Workflows	Type 1	Type 2	Heterogenous Workload (HW)
Workflow deadline ($deadline_W$) (sec)	40-60	80-100	40-60

or arrival rates (λ) of workflows are increased. Deadlines of individual tasks of workflows are calculated using equation 15. In order to compute deadlines of workflow tasks, we put no restriction on the nature of their execution time distributions (general distributions with finite mean and variance) and compute deadlines in a way such that

95% of jobs would execute in time under the calculated deadline. Equation 16 is the cumulative density function of execution time distribution associated with a workflow task. Such bounds or confidence intervals on the execution time can be computed using various techniques such as Chebyshev inequality [13], Monte Carlo approach [14] and Central Limit Theorem [13] or by performing finite integration, if the underlying execution time PDFs are available in analytical forms. Deadline calculation takes care of all possible execution paths in a workflow. $deadline_W$ is the overall workflow deadline for any possible path in a workflow, as shown in table 2. We provide an example for the first task of workflow 2 in figure 2. Mean of the execution time (μ) and coefficient of variation of the execution time (CV) are specified in table 2 with respect to a reference machine. Equation 15 is scaled with reference to $deadline_W$, as it is for the first task of the workflow. Subsequent workflow tasks' deadlines are scaled with reference to the remaining workflow deadline.

$$deadline_1 = \frac{X_1}{X_1 + X_2 + X_3 + X_4 + X_5} deadline_W \tag{15}$$

$$P(0 \leq x \leq X_i) = 0.95 \tag{16}$$

5.2 Results

The job dispatching strategies used with our workload allocation scheme (FF) are round-robin (RR) and random scheduling (RANDOM). These two scheduling strategies are compared with global weighted round-robin (GWRR) and real time information based least-loaded scheduling (RTLL). The GWRR scheme calculates the proportion of workload based on the total processing capacity of each cluster. Hence, higher the total processing power, higher the workload proportion for the cluster. The least-loaded scheme selects those cluster nodes which can satisfy the deadlines of jobs. The workflows don't have any slack period, meaning that they are scheduled without any delay as soon as they are submitted. The main comparison metrics between the schemes are mean execution time of workflows, workflow failures and utilisation of clusters as we increase λ and CV. However we will keep our discussion limited to failures as the main comparison between the schemes is their ability to satisfy QoS requirements.

5.3 Effect of Arrival Rate and Workload Nature

Referring to figures 3 and 4, for low arrival rates, FFRR performs similar to RTLL. However its performance compared to RTLL drops as λ increases. However FFRR and FFRANDOM schemes significantly outperform GWRR. This trends continues, but the advantage gets reducing as arrival rates increase. This can be explained as follows. When arrival rates increase, more work needs to be scheduled in less time and the average response time is an increasing function of arrival rate, as is evident from equation 8 in section 4. Hence failures due to missing deadline assignments increase and as a consequence workflow failures increase. For both low and high CVs, at low arrival rates, FFRR performs similar to RTLL. Referring to figures 5 and 6, for low arrival rates, FFRR performs similar to RTLL. However its advantage over GWRR is significant. Referring to figures 7 and 8, the situation is similar to the above cases. Hence heterogenous workload does not change the behaviour of the schemes.

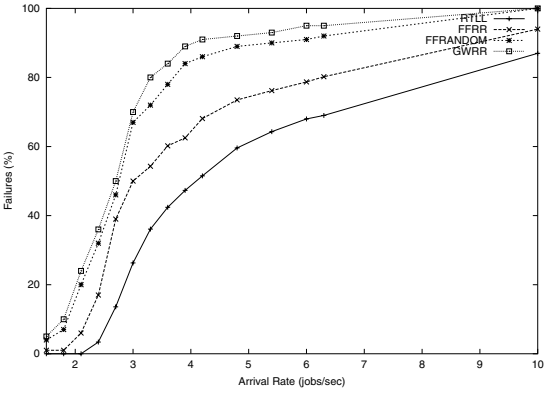


Fig. 3. Failures vs λ, CV = 0.2 (Simulation 1)

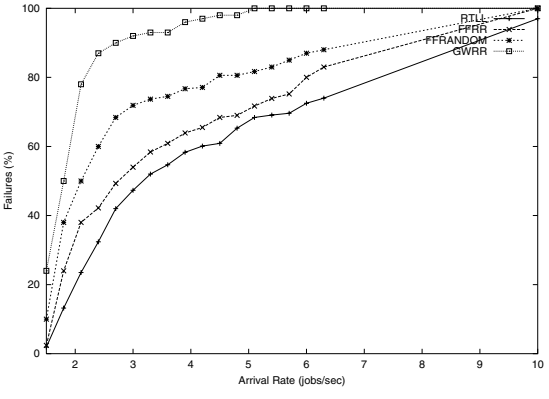


Fig. 4. Failures vs λ, CV = 1.8 (Simulation 1)

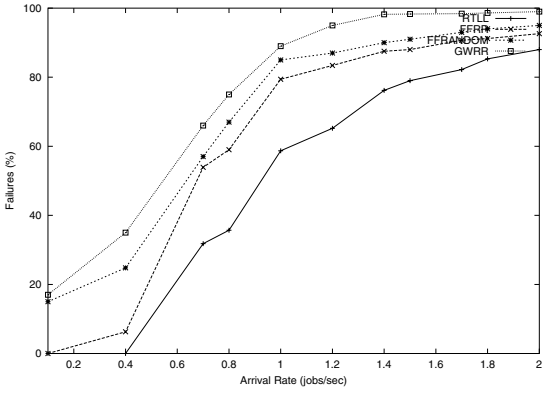


Fig. 5. Failures vs λ, CV = 0.2 (Simulation 2)

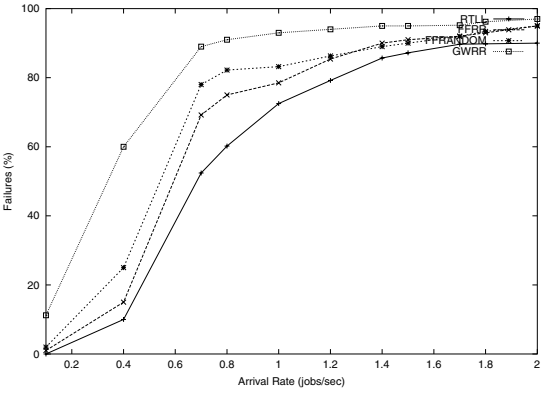


Fig. 6. Failures vs λ, CV = 1.4 (Simulation 2)

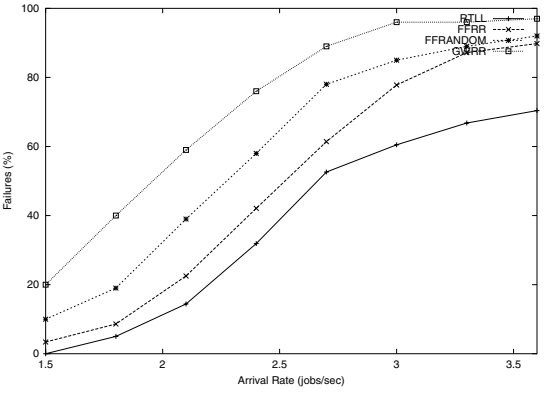


Fig. 7. Failures vs λ, CV = 0.2 (Simulation 3)

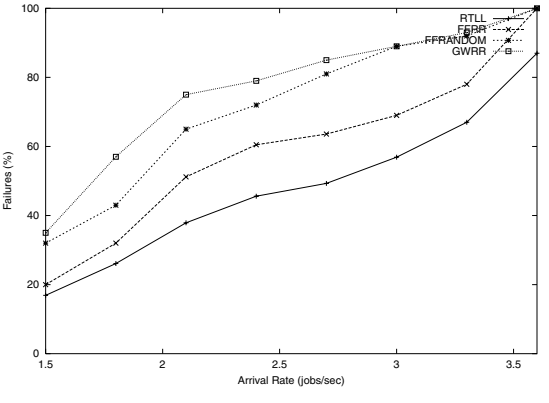


Fig. 8. Failures vs λ, CV = 1.8 (Simulation 3)

5.4 Effect of CV of Execution Time of Workflow Tasks

For both low and high CVs of execution time of jobs, the nature of graphs are similar, however failures increase as CV increases. In case of heterogenous workload, the graphs climb more steeply compared to the case of type 1 workflow. In all cases FFRR significantly outperforms GWRR. This shows that the variability of execution time does not significantly affect the nature of graphs for different schemes. However the advantage of a particular scheme over others reduce as failures reach limiting values asymptotically. As CV is increased, failures increase because workflow jobs take longer time to execute and thus tend to complete near their assigned deadlines or even fail to meet their deadlines.

6 Conclusion and Future Work

The effectiveness of the workload allocation strategy is proved through theoretical analysis. The scheduling schemes combined with workload allocation strategy are also evaluated through experimental simulation. Results confirm that workload allocation strategy combined with scheduling algorithms perform considerably better than the algorithms that do not use these strategies. When the arrival rates are low, the workload allocation technique combined with traditional scheduling strategies perform similar compared to scheduling algorithms based on real time performance information. Workflow and workload nature also don't change the performance of the scheme notably. Moreover execution time variability does not change the performance of schemes significantly for both low are high arrival rates.

As future work we would like to model clusters as $G/G/k$ queues and perform theoretical analysis. The reason behind modelling clusters as $G/G/k$ queues is that $M/M/k$ queues don't often exist in real world situations. We would like to perform experiments with workflows having a slack period, meaning they can wait for some time before getting serviced.

References

1. General Algebraic Modeling System (GAMS). *http://www.gams.com/*.
2. Enabling Grids for E-sciencE (EGEE). *http://www.eu-egee.org/*, 2004.
3. A. Mayer et al. Workflow Expression: Comparison of Spatial and Temporal Approaches. *Workflow in Grid Systems Workshop*, 2004.
4. B. Kao and H. Garcia-Molina. Scheduling Soft Real-Time Jobs over Dual Non-Real-Time Servers. *IEEE Trans. Parallel and Distributed Systems, vol. 7, no. 1*, pages 56–68, 1996.
5. F. Howell et al. SimJava. *http://www.dcs.ed.ac.uk/home/hase/simjava*.
6. G. Nudd and S. Jarvis. Performance-based middleware for Grid computing. *Concurrency and Computation: Practice and Experience*, 2004.
7. K. Chen and L. Decreusefond. Just How Bad is the FIFO Discipline for Handling Randomly Arriving Time-Critical Messages. *Proc. 1995 IEEE International Workshop Factory Communication Systems*, 1995.
8. L. He, S.A. Jarvis, D.P. Spooner, and G.R. Nudd. Optimising Static Workload Allocation in Multiclusters. *Proc. 18th IEEE International Parallel and Distributed Processing Symposium (IPDPS 04)*, 2004.

9. L. He, Z. Han, H. Jin, and L. Pang. DAG-Based Parallel Real Time Task Scheduling Algorithm on a Cluster. *Proc. Seventh International Conference Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, 2000.
10. L. Kleinrock. Queueing Systems. *John Wiley and Sons*, 1975.
11. M. A. Duran and I. E. Grossmann. An Outer-Approximation Algorithm for a Class of Mixed-Integer Nonlinear Programs. *Mathematical Programming*, 36:307–339, 1986.
12. M. Barreto, R. Avila, and P. Navaux. The MultiCluster Model to the Integrated Use of Multiple Workstation Clusters. *Proc. Third Workshop Personal Computer-Based Networks of Workstations*, pages 71–80, 2000.
13. Milton Abramowitz and Irene A. Stegun. Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables. 1972.
14. N. Metropolis and S. Ulam. The Monte Carlo Method. *Journal of the American Statistical Association*, 1949.
15. N.G. Shivaratri, P. Krueger, and M. Singhal. Load Distribution for Locally Distributed Systems. *Computer, vol. 8, no. 12*, pages 33–44, 1992.
16. O. Aumage. Heterogeneous Multi-Cluster Networking with the Madeleine III Communication Library. *Proc. 16th International Parallel and Distributed Processing Symposium (IPDPS 2002)*, 2002.
17. R. Leslie and S. McKenzie. Evaluation of Load Sharing Algorithms for Heterogeneous Distributed Systems. *Computer Communications*, 1999.
18. S.A. Banawan and N.M. Zeidat. A Comparative Study of Load Sharing in Heterogeneous Multicomputer Systems. *Proc. 25th Annual Simulation Symposium*, 1992.
19. W. Zhu and B. Fleisch. Performance Evaluation of Soft Real-Time Scheduling on a Multicomputer Cluster. *Proc. 20th International Conference Distributed Computing Systems (ICDCS 2000)*, pages 610–617, 2000.
20. Xuehai Zhang and Jennifer M. Schopf. Performance Analysis of the Globus Toolkit Monitoring and Discovery Service, MDS2. In *Proceedings of the International Workshop on Middleware Performance (MP 2004)*, Apr. 2004.
21. X.Y. Tang and S.T. Chanson. Optimizing Static Job Scheduling in a Network of Heterogeneous Computers. *Proc. 29th International Conference on Parallel Processing*, pages 373–382, 2000.

Managing Multiple Isolation Levels in Middleware Database Replication Protocols*

Josep M. Bernabé-Gisbert, Raúl Salinas-Monteagudo, Luis Irún-Briz, and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
{jbgisber, rsalinas, lirun, fmunyoz}@iti.upv.es

Abstract. Many database replication protocols have been designed for guaranteeing a serialisable isolation level, since it is appropriate for almost all applications. However, it also requires a tight coordination among replicas and might generate high abortion rates with some workloads. So, other isolation levels have also been considered, such as snapshot isolation and cursor stability, but none of the previous works has proposed an overall support for more than one isolation level at the same time. This paper explores such a research line.

1 Introduction

Many data replication protocols have been published for years [1,2,3], and they have always been centred on a single isolation level. Indeed, when multiple isolation levels have been presented [4], a separate protocol has been designed for each of them. There is no problem with this approach, since it makes possible a thorough description, discussion or justification for each protocol. However, applications may often require that their transactions were executed in different isolation levels, mainly for improving the access time of such transactions that tolerate reading non-strictly-consistent data. This necessity of managing multiple isolation levels is a main issue for database applications, and has been even included as part of several "*standard*" benchmark applications, such as the one defined in the TPC-C [5] specification. In such benchmark, its New-Order, Payment, Delivery and Order-Status transactions require the ANSI *serialisable* level, and the same set of transactions requires that other transactions accessing the same data (besides the Stock-Level one, that is also included in the benchmark) use the *repeatable read* level, whilst its Stock-Level transaction only demands the *read committed* level. Many applications follow similar patterns on their sets of transactions.

Any serious centralised *database management system* (DBMS, on the sequel) is able to manage without problem multiple isolation levels at a time (i.e. for several concurrent transactions), but a database replication middleware is faced with some inconveniences for providing such service. Mainly, there is no trivial way of coordinating different replication protocols, each one providing support for a single isolation level.

* This work has been partially supported by the Spanish grants TIC2003-09420-C02 and TIN2006-14738-C02.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 511–523, 2006.
© Springer-Verlag Berlin Heidelberg 2006

As a result, when such applications must be managed, there are only three options for dealing with them. The first one is to discard the modern database replication techniques, following a distributed locking approach for concurrency control. The rules for providing the most important isolation levels have already been specified for locking techniques [6], and are easily implementable in distributed systems with distributed locks. However, distributed locking has proven to show a poor performance when compared with replication techniques based on total ordered write-set propagation [3]. The second approach consists in selecting a set of modern protocols with similar techniques and different isolation levels, defining from scratch the rules to be followed when different levels must be combined. This may be achieved when such protocols use similar solutions for the most important parameters that define a replication protocol [7]: server architecture, replica interaction, and transaction termination. The last option consists in supporting a single isolation level –the strictest one being needed–, thus requiring that all transactions were executed using such level. This leads to poor performance or higher abortion rates for those transactions that would have tolerated a more relaxed isolation level.

This paper describes a general scheme for designing replication protocols that support multiple isolation levels. Although there are multiple levels that could have been supported, this first solution only considers four basic alternatives that are quite similar to the ANSI standard levels, according to the generalised definitions proposed in [8].

The rest of the paper is structured as follows. Section 2 presents our system replication model. Section 3 outlines the supported isolation levels and how they have been implemented in previous replication protocols. Section 4 describes our solution, whilst section 5 compares it with other related work. Finally, section 6 concludes the paper.

2 System Model

We assume a fully replicated database; i.e., each node holds a replica of our database. Each system node has a local DBMS that is used for locally managing transactions, and that provides the mechanisms needed for ensuring the standard ANSI isolation levels. On top of the DBMS a middleware is deployed in order to provide support for replication. This middleware also has access to a group communication service that should support atomic multicast [9] (or uniform atomic multicast if failures are considered). Our solutions might be also used in non middleware-based systems, but this requires at least a minimal modification of the DBMS core, and such extension depends on the DBMSes being considered. We do not describe such dependencies in this paper, so our discussion is better tailored for middleware solutions.

The replication model being used is *read one, write all available* (ROWAA, on the sequel), since in almost all replication protocols only the transaction write-sets are propagated. The comparison made in [3] also proves that this behaviour provides better performance than any other that requires read execution in all replicas.

3 Isolation Levels

Many current relational DBMSes support the standard ANSI SQL-92 isolation levels. However, the definitions given in such standard are not enough precise, as they were

criticised in [6]. In that paper, its authors distinguished between strict interpretations of the *phenomena*[1] discussed in the standard, and loose interpretations, showing with some examples that with a strict interpretation some non-desired anomalies were possible in each isolation level. As a result, the standard specification must be understood using the loose interpretations outlined in [6] that generate stricter levels of isolation. Some traditional implementations based on locks already supported such loose interpretations, but others did not. Thus, some DBMSes using multi-version concurrency control (MVCC, for short) had followed the strict phenomena interpretations. Consequently, they only provided a *snapshot* isolation level (as defined in [6]) when they were asked for a *serialisable* one.

Unfortunately, both the loose phenomena interpretation and the lock-based concurrency control proscribed some transaction executions that were perfectly legal for the required isolation levels. Adya et al. [8] detected such problems and specified again the isolation levels. Their specifications are more precise than those presented in [6] and also implementable with optimistic concurrency control (and this is the most common in replicated systems, since transactions are generally allowed to proceed until they request their commit and get validated or certified).

So, in order to be complete, we provide on the sequel a summary of the phenomena definitions given in [8] that should be proscribed in some of the standard isolation levels:

G0 (*Write cycles*): A history H exhibits phenomenon G0 if DSG(H) contains a directed cycle consisting entirely of write-dependency edges.
In this definition, DSG(H) is a *direct serialisation graph* [8] based on direct conflicts between committed transactions. Additionally, a write dependency occurs when one transaction overwrites a version written by another transaction.

G1a (*Aborted reads*): A history H shows phenomenon G1a if it contains an aborted transaction T1 and a committed transaction T2 such that T2 has read some object modified by T1.

G1b (*Intermediate reads*): A history H shows phenomenon G1b if it contains a committed transaction T2 that has read a version of object x written by transaction T1 that was not T1's final modification of x.

G1c (*Circular information flow*): A history H exhibits phenomenon G1c if DSG(H) contains a directed cycle consisting entirely of dependency edges.
In this phenomenon definition, a dependency edge is either a write dependency (see G0 description) or a read dependency. A read dependency arises when a transaction reads some items written by another transaction, or when the results of a transaction read (using a predicate) are modified by a write operation made by another transaction (including value changes, as well as element additions or removals in such results). The results in such predicate-based queries are all items accessed, plus their corresponding *truth degree* for the predicate, even if they do not match such predicate. Those items that match the predicate are added to the history as separate individual reads. So, the write operations that include or remove elements in a predicate read are those that inserted or deleted such items in or from their respective tables.

[1] The term *phenomenon* refers to consistency anomalies that should be avoided when transaction isolation is enforced.

G2 (*Anti-dependency cycles*): A history H exhibits phenomenon G2 if DSG(H) contains a directed cycle with one or more anti-dependency edges.

Informally, an anti-dependency arises when a transaction overwrites a version observed by some other transaction.

When the anti-dependencies arise between transactions that do not use predicate-based reads, a *G2-item* phenomenon occurs. In the general case (i.e., with the *G2* phenomenon) both kinds of read operations are considered (predicate-based and item-based).

These definitions match respectively the original P0, P1, P2 (equivalent to G2-item) and P3 (equivalent to G2) phenomena definitions of the ANSI standard. However, P1 was decomposed in three different G1 subcases in order to eliminate the problems detected in the loose interpretations proposed by [6]. Consider also that G1 implicitly includes the G0 phenomenon, so if a level proscribes G1 it also proscribes G0. With these phenomena definitions, Adya et al. specify some portable levels of isolation that we summarise in table 1. We use these specifications in the following sections in order to build a set of rules that might be used for defining general replication protocols able to manage multiple isolation levels.

Table 1. Portable ANSI isolation levels

Level	*Disallowed phenomena*	*Equivalent ANSI level*
PL-1	G0	READ UNCOMMITTED
PL-2	G1	READ COMMITTED
PL-2.99	G1, G2-item	REPEATABLE READ
PL-3	G1, G2	SERIALISABLE

4 A General Replication Protocol

There are many ways of writing a database replication protocol, since there are some parameters that define how such protocol should behave. Thus, in [7] three parameters of this kind were identified: server architecture, server interaction and transaction termination. Each one of these parameters can take two different values, generating eight different classes of protocols. A replication protocol supporting multiple isolation levels should be able to match any protocol in all these classes. Unfortunately, there are big differences among such classes, and it would be difficult to provide a single principle easily adaptable to all of them.

For instance, the *server architecture* parameter distinguishes between protocols based on a primary server where all transactions should be forwarded, and protocols that allow the execution of transactions in any site (called *update everywhere* replication). Regarding concurrency control and isolation, the primary server approach does not imply any problem, since the execution of transactions is fully centralised and we may rely on the local concurrency control mechanisms in such primary copy; i.e., the protocols we are looking for are trivially implementable in this kind of replication since only one replica should take care of concurrency control, and its semantics can be directly driven by the underlying DBMS.

Besides this, other classes can be easily discarded due to other problems not related with isolation, but with other requirements such as performance. For instance, linear interaction (one of the alternatives for the server interaction parameter) implies extremely expensive overheads on communication among replicas, and complicates a lot the recovery subprotocols. So, it is commonly discarded in the general case.

As a result of this, only two of the original eight classes identified in [7] should be considered in this paper: those based on an *update everywhere* server architecture, with *constant* server *interaction* and with either *voting* or *non-voting* transaction termination.

So, once identified the target protocol classes to be managed by our general solution, let us see which implementation choices we assume and how a general protocol can be defined, also proving how is it able to avoid each of the phenomena described in [8].

4.1 Protocol Implementation Features

There have been multiple database replication protocols in the *update everywhere* server architecture with a *constant* server *interaction* [4,10,11,12]. Many of them share the following characteristics, proving to be extremely adequate for replication purposes. Thus, we will take them as a basis for designing our general protocol:

- Since they belong to the update everywhere server architecture, transactions can be initiated in any replica. There is no special replica that centralises transaction management.
- As they also belong to the constant interaction class, only a constant number of messages are exchanged among replicas. In the common case, such messages are used for propagating the updates, and they are needed once the commit has been locally requested in the initiating replica. Although other solutions are possible, we will limit our discussion to protocols that propagate the transaction data at the end of each transaction; i.e., when the application has locally requested the commit.
- Write-set (and, in some cases, read-sets [12]) propagation is made using an *atomic multicast*; i.e., a multicast with message delivery in total order. This ensures that all replicas see the same sequence of write-sets (and, if needed, read-sets); i.e., the same sequence of transactions.
- The underlying DBMS provides support for the isolation level being requested by the user transactions. Thus, local transactions can be managed by the underlying DBMS, and the middleware must ensure that the mix among remote and local transactions also follows the requested isolation levels.

Taking these features as a basis, the design of a database replication protocol is reduced to check for conflicts between local transactions and write-sets being delivered, or between those write-sets. Additionally, two schemes for such checking are possible, depending on the transaction termination alternative being chosen [7]: either a voting phase is needed in the transaction termination, or all replicas behave deterministically in the certification phase and all arrive to the same decision without needing any explicit coordination. But protocols based on voting can be divided in two different subclasses: those that are symmetrical, requiring a vote by every replica (for instance, in order to cope with unilateral abortions [13] or other sources of non-determinism), or others that

rely on a delegate server[2], who imposes its decision to the rest of replicas (this approach is referred to as *weak voting replication* in [3]).

Between these three approaches for terminating transactions, we choose only the weak voting replication approach, since the other two have the following problems:

- *Non-voting termination*. In this case, if the *serialisable* isolation level has to be supported, read-sets must be propagated [13]. Although there are some techniques that allow read-set propagation with minimal costs [12], read-set collection can be a problem for long transactions.
- *Symmetrical voting termination*. The communication needs of this voting phase, plus those already paid for total order write-set delivery generate an overall communication cost similar to a 2PC. This scheme might be supported if a non-atomic multicast is used, such as in the protocols described in [14], but with the scheme outlined in this section its costs are too high to consider it appropriate.

Although these two approaches will not be the focus of this paper, the solution described in the following sections might be easily adapted to both of them. In all approaches a validation phase is needed, and the issues being considered in these validations are not too different among these approaches.

4.2 A General Scheme

Our general scheme for supporting multiple isolation levels is based on the following principles:

- If multiple isolation levels should be supported, a protocol for the strictest isolation level –among those to be supported– has to be selected.
- When a transaction is started, its intended isolation level should be requested to the underlying DBMS.
- When a transaction reaches the commit phase, and its write-set (and, in some cases, read-set) is propagated, its isolation level identifier has to be included into such propagation message.
- The validation step needed in the replication protocol for deciding whether a transaction must commit or abort has to consider the isolation levels of all the transactions being checked. The rules to check between transactions that have requested different isolation levels have to consider the phenomena to be proscribed by such isolation levels.

These principles are general enough to be applied to any transaction termination approach (i.e., weak voting, symmetrical voting, and non-voting cases). In this paper, such scheme will be applied to the weak voting replication approach. So, this kind of database replication must be considered as a case study for our general scheme.

A database replication protocol based on weak voting replication consists in the following steps [3]:

[2] The delegate server is the replica that has initiated the particular transaction.

1. When a delegate database server DS_d receives a transaction T from a client C, it executes the transaction but delays its write operations.
2. When client C requests the transaction commit, the transaction write-set is propagated to all replicas using atomic broadcast. Note that if a transaction has an empty write-set (i.e., it is a read-only transaction) no broadcast is needed and it immediately commits.
3. When such write-set message is delivered, the delegate server determines if conflicting transactions have been committed.
4. If so, transaction T must be aborted. Otherwise, it should be committed. Depending on the result of this validation, the replica DS_d uses a reliable broadcast to propagate this result.
5. Concurrently with these two last steps, the other replicas have received the same write-set and have locally applied it. Once they receive the validation result, they take the appropriate action (either to abort or to commit transaction T).

This protocol is able to provide a *serialisable* isolation level, but the key for this resides in its step number 3, where the write-set is validated and a result for each transaction is decided. Depending on the rules being used for determining "conflicting" transactions other isolation levels can be obtained.

For applying our general scheme, we only need to extend minimally this sample algorithm in order to:

a) Extend its step 1, requesting to the underlying DBMS the appropriate level.
b) Extend its step 2, including the isolation level of such transaction as an additional field into the write-set message.
c) Adapt its step 3, using the appropriate conflict checking rules for each level.

This last topic deserves further explanation and is discussed on the sequel.

4.3 Avoiding General Phenomena

In this section, we will show how the general phenomena presented in section 3 can be proscribed using some concurrency control techniques and validation checks in the protocol outlined above. To begin with, let us start with the mechanisms needed for guaranteeing the isolation level PL-3, and later discussing how the other levels (PL-2.99, PL-2, and PL-1, respectively) can be ensured. In all these variants, read-set propagation is not needed since read accesses are only checked against write-sets (from either local or remote transactions) in the delegate replica where such transactions have been started.

Portable Level PL-3. This portable level is almost identical to the ANSI *serialisable* level. It requires that both G2 and G1 phenomena were proscribed.

Using traditional locking techniques, this isolation level needs long read and write locks. In a replicated environment, these locks should be combined with the total order being guaranteed by the atomic broadcast.

An example of database replication protocol that uses the weak voting replication approach ensuring a *serialisable* level is the SER protocol of [4]. This solution also

uses a lock-based concurrency control, requesting long locks in the delegate server for both kinds of accesses (reads and writes), and requesting also write locks when the write-set is delivered in remote replicas. As a result of this, its validation procedure distinguishes the following actions:

1. The write-set application may get blocked in non-delegate replicas if the requested locks conflict with the locks already acquired by other transactions that have been previously delivered following the total order of the atomic broadcasts. Thus, such write-set application simply waits for the completion of such conflicting transactions, and no rollback is needed in this case.
2. Otherwise, if such lock request collides with some local read locks that belong to transactions whose write-sets have not been delivered, such local transactions are aborted.

Thus, in order to forbid phenomenon G2, we must ensure that no cycle with at least one anti-dependency edge might be created in any execution of this protocol. Recall that T1 has an anti-dependency on T2 if T1 overwrites an item (or the result of a predicate evaluation) read by T2. In this protocol cycles are prohibited, since the total order delivery ensures that all transactions are sequentially ordered and thus, it is impossible that the same transaction initiates and terminates a cycle of dependencies (it will be either the first or the last in such order, but not both since the local concurrency control in all replicas also prevents such kind of cycles among local transactions).

Supose that a node N_i is trying to apply T_i's write-set WS_i. Validation action 1 ensures that a WS_i is never applied before any previous conflicting delivered transactions because WS_i will be blocked until these transactions commit. Additionally, all not yet delivered local transactions with read locks on items accessed by T_i never commit before T_i because validation action 2 would abort them. Both actions combined ensure that the destination transaction for every dependency or anti-dependency edge commits after its source transaction. This implies a sequential committing order, and justifies the avoidance of phenomenon G2.

In a similar way, G1 is avoided since G1c is also proscribed due to the total order delivery, introducing a sequential order of transactions that prevents cycles from appearing in the DSG(H) of any history H. Moreover, the use of local long write locks avoids phenomena G1a and G1b. Thus, G1a (aborted reads) is avoided because due to the long write locks, it is impossible that a transaction T2 would have read an item previously written by a transaction T1 that finally had aborted. The same happens with G1b (intermediate reads).

If, instead of a lock-based concurrency control other local concurrency control approaches were used similar validation actions would be needed. For instance, with MVCC, the validation action 1 would have had the same behaviour, since write conflicts lead to blocking with such kind of concurrency control. On the other hand, the validation action 2 would have had a difficult management with this kind of concurrency control, since no locks are requested for reading. As a result, local read operations should be translated into SELECT FOR UPDATE statements in order to detect such read-write conflicts and a mechanism such as the one described in [15] would be needed for dealing with such kind of conflicts, leading to the abortion of these local transactions.

As it has been explained for lock-based concurrency control, this solution proscribes both G2 and G1 phenomena. Both G2 and G1c are prohibited by the total order being used for write-set delivery, whilst both G1a and G1b are trivially avoided by the underlying MVCC, since the versions being read by each transaction have been generated by transactions already committed (intermediate versions are always private for the transaction that has generated them, when a *serialisable* isolation level is requested in a MVCC system).

Portable Level PL-2.99. This second portable level (PL-2.99) is almost equivalent to the ANSI *repeatable read* isolation level. For ensuring it, in lock-based concurrency control long locks are used for write and item-read operations, but only short locks when the read operations use a predicate. If we plan to use an underlying DBMS with this kind of concurrency control, we may use the same validation actions than we described for PL-3 –but considering that now predicate reads only need short locks and, as a result, will not get aborted by validation action 2–. Since transactions that need PL-2.99 have requested the *repeatable read* isolation level to the underlying DBMS, conflicts among PL-3 and PL-2.99 writing transactions will be correctly managed by such DBMS. In case of conflicts between remote write-sets and local reading transactions, the middleware will be able to detect such conflicts using the mechanisms outlined in [15]; i.e., reading one of the system-catalogue tables that records those transactions that have been blocked due to conflicts with other transactions.

As a result of this, no modification over the solution already described for PL-3 is needed for achieving PL-2.99 at the middleware level. Additionally, the justification of the proscription of the G2-item and G1 phenomena is identical to those already given above for PL-3.

If MVCC is used, there is no way to allow phenomena G2 for predicate-based reads; i.e., allowing anti-dependency edges that overwrite predicate reads. Some DBMS based on MVCC are not able to provide an ANSI *repeatable read* isolation level: PostgreSQL [16] is an example. This kind of concurrency control ensures that each transaction gets item versions that correspond to the moment when such transaction was started. As a result of this, a write operation generates a new version for every updated item, but such version can not be accessed by concurrent transactions. So, the isolation achieved with this concurrency control technique for read accesses is more or less equivalent to using long read locks in a lock-based technique. Thus, level PL-2.99 is not achievable with this kind of concurrency control. On the other hand, this kind of concurrency control easily provides the *snapshot* isolation level [6] that shares some of the characteristics of this PL-2.99 level but that is not equivalent to it.

Portable Level PL-2. In this portable level (almost equivalent to the ANSI *read committed* level), phenomenon G2 is allowed, but G1 is still proscribed. So, anti-dependency edges may be present, being able to close dependency cycles among a given set of transactions. In an implementation based on locks this level only requires short locks for read accesses, and long locks for writes.

Regarding our sample protocol described for the PL-3 level, in this case validation action 2 can be removed since the application of a remote write-set would not abort any local transaction. As a consequence, no abortion is generated in such validation

actions and this means that the reliable broadcast needed in step 4 of the sample protocol outlined in section 4.2 will not be needed by transactions that belong to the PL-2 and PL-1 isolation levels.

If the underlying DBMS supports this PL-2 level, both G1a and G1b phenomena are proscribed, since these phenomena are caused by read accesses and they can only be local in our sample general protocol. Phenomenon G1c should also be proscribed. To this end, no cycle of dependency edges should be allowed by our protocol. This is easily ensured, since validation action 1 ensures that write-dependencies can only be established in the order being imposed by the atomic delivery of write-sets, and this prevents the appearance of write-dependency cycles. Read-dependencies can be locally present in some replicas (in those where each transaction had its delegate server) but they would not be able to close any cycle. Otherwise, a single transaction would have read some information from a write-set that occurs after it in the write-set total order delivery, and this is impossible (since local concurrency control mechanisms prevent a transaction from reading something that has not yet been committed).

As stated above, phenomenon G2 should be allowed. So, a history like the following one should be permitted (it follows the notation proposed in [8]):

$$H: r_1(Weight{>}50{:}x_0{,}60;y_0{,}51)\ r_1(x_0{,}60)\ r_2(y_0{,}51)\ w_2(y_2{,}50)\ c_2\ r_1(y_2{,}50)\ c_1$$

In such sample, transaction T1 gets all items with a weight greater than 50 and two items are returned, x and y. Concurrently T2 updates y weight, setting it to value 50. Finally, when T1 gets y's data it recovers a 50 value that does not match the predicate being used. So, we have a T1 read-dependency on T2 and a T2 anti-dependency on T1. This situation is trivially allowed by our general protocol since read-only transactions are not broadcast to all replicas, and in this case T1 –a read-only transaction– has been allowed to commit by the local concurrency control on its delegate replica. Additionally, T2 is broadcast and committed without problems in all database replicas.

In a general solution, with other validation techniques for its PL-3 level, the checks being made for write-set collisions should be maintained in this PL-2 level, but those checks associated to write-read conflicts can be eliminated if a local concurrency control able to provide PL-2 is used.

Portable Level PL-1. Finally, portable level PL-1 (similar to *read uncommitted*) only proscribes phenomenon G0; i.e., write-dependency cycles. As already discussed in the previous case, write-dependency cycles are avoided if the write-set delivery order is respected when such write-sets are applied on each replica. Again, as in the previous case, the specific validation action 2 is not needed. However, now the underlying DBMS needs to enforce locally only the PL-1 level and this places some restrictions on write accesses but never on reads. Due to this type of read management, read-dependencies may arise, allowing thus the occurrence of phenomena G1a, G1b and G1c.

The general solution to this isolation level is the same already stated for PL-2: to take care only for write-write conflicts when write-sets are delivered. The single difference with that level is that now some local concurrency control support specific for a PL-1 level is assumed. Note that not all MVCC DBMSes provide a so relaxed isolation level. For instance, PostgreSQL [16] is able to provide support for *read committed* (PL-2) and its *serialisable* (PL-3) (actually, *snapshot*) levels, but not for *read uncommitted* (PL-1)

nor for *repeatable read* (PL-2.99). The same happens in Microsoft SQL Server 2005 [17] when it is configured for using optimistic concurrency control techniques.

Summary. This section has shown that dealing with multiple isolation levels in a middleware-layer replication protocol is feasible. To this end, a good replication protocol for the strictest isolation level has to be chosen and local support for all the intended isolation levels should be present in the underlying DBMS. If all these requirements can be coped with, the concurrency control checks will be quite similar for all these isolation levels, but part of them are lost in the looser levels being supported. Thus, when a write-set arrives, the receiving replica must only apply the checks associated to the isolation level of such incoming write-set. Note also that some non-standard isolation levels might require other techniques for avoiding their proscribed phenomena (e.g., in case of the *snapshot* or *cursor stability* levels), but such cases will be studied in further works to be completed in the near future.

Figure 1 shows the resulting database replication protocol supporting the standard isolation levels that we have generated taking the SER protocol of [4] as its basis.

1. When a transaction T_i starts, set its isolation level on the local DBMS.
2. When T_i is locally terminated:
   2.1. If it is read-only, it directly commits.
   2.2. Otherwise, get its write-set (WS_i) and its isolation level (IL_i) and broadcast them in total order to all replicas.
3. Upon (WS_i,IL_i) delivery:
   3.1. For each operation on WS_i:
      a) If $IL_i >$ PL-2 and there is a read-write conflict with a local transaction T_j that has not delivered its write-set, abort T_j.
      - If T_j has broadcast (WS_j,IL_j) and $IL_j >$PL-2, broadcast abort(T_j).
      b) If there is any other conflict with another local transaction T_j, ensure that WS_j and WS_i are applied in their delivery order.
   3.2. Apply WS_i locally.
   3.3. If T_i is a local transaction and $IL_i >$PL-2, broadcast commit(T_i).
   3.4. If $IL_i <$PL-2.99, commit T_i.
4. Upon commit(T_i) or abort(T_i) delivery, commit or abort T_i, depending on the received message.

Fig. 1. A sample of general replication protocol

5 Related Work

Most current database replication protocols aim to provide support for only the *serializable* [3,10] or *snapshot* isolation levels [11], since they are needed by a wide variety of applications. However, there have also been some works that have studied multiple levels of isolation, either providing protocols for each of them [4,12] or by specifying new definitions of such levels [6,8]. But none of these works has supported more than one isolation level in a single replication protocol. As a result, they have designed good solutions for a single level but they can not be merged easily into a single protocol, since such solutions are specifically tailored for their target level.

Despite this, there have been some attempts to support multiple isolation levels in a single protocol. This was one of the aims in the GlobData project and some initial solutions were provided in [18]. But the isolation levels defined in GlobData were not the ANSI standard ones, since GlobData was a system with an object-oriented interface able to provide an object-oriented replicated database using relational database replicas, and for those systems, there were considered another set of behaviours. So, such initial solution is not comparable to the one discussed in this paper.

Other papers deserve special attention although their objectives are not exactly the same as ours. In [12] two new protocols are described. The first one is an evolution of the original *Database State Machine* (DBSM) approach [13] that provides *snapshot* isolation, while the other uses some rules that are similar to those of the original DBSM but it is able to guarantee serialisability without transferring read-sets. The latter manages conflict classes (or logical sets) and introduces dummy writes that are able to simulate the read-sets. Unfortunately, these two protocols are not based on the same principles (the first one uses an *update everywhere* server architecture with non-voting termination, whilst the second one uses a primary copy server architecture) and, as a result of this, they will be difficult to merge in a single protocol supporting both isolation levels. On the other hand, it analyses two of the most used isolation levels.

Similarly, in [4] two different protocols were provided for supporting such two isolation levels. Moreover, in [4] other non-standard isolation levels (*cursor stability*, for instance) were also supported by other protocols. However, although all of them share similar architectures (update everywhere server architecture and constant interaction), nothing is said about merging all levels in a single protocol.

6 Conclusions

This paper has presented a general scheme for designing middleware database replication protocols supporting multiple isolation levels. It is based on progressive simplifications of the validation rules used in the strictest isolation level being supported, and on local (to each replica) support for each isolation level in the underlying DBMS.

Such scheme provides a uniform management for all isolation levels, needing minimal extensions to the original database protocol (the one initially designed for the strictest level). This support for multiple isolation levels is specially fruitful for those applications that manage multiple kinds of transactions, since they get an improved performance for those transactions able to run with the loosest isolation levels. Without the described general protocol, an application of this kind would have used a single replication protocol supporting the strictest needed isolation level, and this would have penalised its performance, or would increase the abortion rate of the most relaxed transactions.

References

1. Bernstein, P.A., Goodman, N.: An algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. on Database Sys. **9** (1984) 596–615
2. Carey, M.J., Livny, M.: Conflict detection tradeoffs for replicated data. ACM Trans. on Database Systems **16** (1991) 703–746

3. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. on Knowledge and Data Engineering **17** (2005) 551–566
4. Kemme, B., Alonso, G.: A new approach to developing and implementing eager database replication protocols. ACM Transactions on Database Systems **25** (2000) 333–379
5. Transaction Processing Performance Council: TPC benchmark C - standard specification (2005) Revision 5.6. Downloadable from: http://www.tpc.org/.
6. Berenson, H., Bernstein, P., Gray, J., Melton, J., O'Neil, E., O'Neil, P.: A critique of ANSI SQL isolation levels. In: SIGMOD Intnl. Conf. on Management of Data, San José, CA, USA (1995) 1–10
7. Wiesmann, M., Schiper, A., Pedone, F., Kemme, B., Alonso, G.: Database replication techniques: A three parameter classification. In: 19th Symposium on Reliable Distributed Systems. (2000) 206–217
8. Adya, A., Liskov, B., O'Neil, P.: Generalized isolation level definitions. In: IEEE Intnl. Conf. on Data Engineering, San Diego, CA, USA (2000) 67–78
9. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In Mullender, S., ed.: Distributed Systems. 2nd edn. ACM Press (1993) 97–145
10. Pedone, F., Guerraoui, R., Schiper, A.: The database state machine approach. Distributed and Parallel Databases **14** (2003) 71–98
11. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In: SIGMOD Conference. (2005) 419–430
12. Zuikeviciute, V., Pedone, F.: Revisiting the database state machine approach. In: VLDB Workshop on Design, Implementation and Deployment of Database Replication, Trondheim, Norway (2005)
13. Pedone, F.: The Database State Machine and Group Communication Issues. PhD thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland (1999)
14. Armendáriz-Íñigo, J.E.: Design and Implementation of Database Replication Protocols in the MADIS Architecture. PhD thesis, Univ. Pública de Navarra, Pamplona, Spain (2006)
15. Muñoz, F.D., Pla, J., Ruiz, M.I., Irún, L., Decker, H., Armendáriz, J.E., González de Mendívil, J.R.: Managing transaction conflicts in middleware-based database replication architectures. In: 25th IEEE SRDS, Leeds, U.K. (2006)
16. PostgreSQL Global Development Group: PostgreSQL 8.1: Concurrency control (2006) Accessible in URL: http://www.postgresql.org/docs/8.1/static/mvcc.html.
17. Microsoft Corp.: Choosing row versioning-based isolation levels (2006) Microsoft Developer Network. Accessible in URL: http://msdn2.microsoft.com/en-us/library/ms188277.aspx.
18. Muñoz, F.D., Irún, L., Galdámez, P., Bernabéu, J., Bataller, J., Bañuls, M.C.: Consistency protocols in GlobData. In: X Jornadas de Concurrencia, Jaca, Spain (2002) 165–178

Proof and Evaluation of a 1CS Middleware Data Replication Protocol Based on O2PL*

J.E. Armendáriz-Iñigo[1], F.D. Muñoz-Escoí[1], J.R. Garitagoitia[2], J.R. Juárez[2], and J.R. González de Mendívil[2]

[1] Instituto Tecnológico de Informática, Camino de Vera s/n 46022 Valencia, Spain
{armendariz, fmunyoz}@iti.upv.es
[2] Universidad Pública de Navarra, Campus de Arrosadía s/n 31006 Pamplona, Spain
{joserra, jr.juarez, mendivil}@unavarra.es

Abstract. Middleware data replication techniques are a way to increase performance and fault tolerance without modifying the internals of a DBMS. However, they introduce overheads that may lead to poor response times. In this paper a modification of the O2PL protocol is introduced. It orders conflicting transactions by using their priority, instead of the total order obtained by an atomic multicast. Priorities are also used to avoid deadlocks. For improving its performance, it does not use the strict 2PC rule as O2PL does. We provide a formal correctness proof of its 1-Copy Serializability (1CS). This protocol has been implemented, and a comparison with other already implemented protocols is also given.

1 Introduction

Database replication is an attractive way for increasing the performance and fault tolerance of applications, but they pay a price for maintaining data consistency. Traditionally, replication has been achieved modifying the Database Management System (DBMS) internals, such as [1,2,3] but this solution is not portable among different DBMS vendors. The alternative approach is to deploy a middleware architecture that creates an intermediate layer that features data consistency, being transparent to the final users. However, one drawback of the middleware approach is that the replication module usually reimplements many features provided by the DBMS. Besides, the database schema has to be extended with standard database features, such as functions, triggers, stored procedures, etc. [4], in order to manage additional metadata that eases replication. This alternative introduces an overhead that penalizes performance but permits to get rid of DBMSs' dependencies. Hence, the goal is to design a system that penalizes performance as less as possible, and that becomes portable to different DBMSs.

The strongest correctness criterion for database replication is 1CS [1] that implies a serial execution over a logical data unit although there are many physical copies. In [4] a middleware architecture is introduced providing 1CS by way of the total order multicast featured by a Group Communication System (GCS) [5,6]. The total order multicast is used so as to determine the order in which transactions are executed on the system. This is an interesting approach since transactions do not have to wait for applying the

* Work supported by the Spanish MEC grants TIC2003-09420-C02 and TIN2006-14738-C02.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 524–537, 2006.
© Springer-Verlag Berlin Heidelberg 2006

updates at the rest of nodes in order to commit, as the 2PC rule states, increasing its performance. However, to rely on these strong GCS primitives is a high price to pay in environments where conflicts are rare, due to the latency and extra message rounds introduced by the total order multicast [6,3].

O2PL [2] was one of the first replication protocols that followed the Read One Write All Available (ROWAA) approach [1]. Transactions are firstly executed at their closest node (or *master node*, hereafter) and $updates$ are propagated to the rest of nodes without any ordering assumption. Updates reception at the rest of nodes starts a remote transaction requesting a *copy-lock* for applying the updates. This lock behaves like a write lock does, but it is used to prevent deadlocks with local transactions. Despite of this, a *snoop process* is still needed to detect and resolve distributed deadlocks. Once updates are applied at a replica, it sends a message to the master node saying it is $ready$ to commit. Meanwhile, the master node collects all $ready$ messages coming from the rest of replicas. At the time when all nodes have answered, the master node commits and multicasts a $commit$ message to all replicas. Therefore, O2PL is a 2PC protocol since it waits for the updates application at all available replicas before committing a transaction.

Here, we propose an evolution of O2PL [2] adapted to our MADIS architecture [7] called Enhanced Replication Protocol (ERP). Using MADIS, the concurrency and replica control may be split into two levels: the underlying DBMS at each node providing serializable isolation level whilst the middleware layer is in charge of data consistency. Hence, no specific DBMS feature has to be re-implemented at the middleware layer. ERP only needs a reliable multicast as the communication mechanism among replicas [5]. We have changed the 2PC philosophy of O2PL: a transaction does not wait for applying the updates at the rest of nodes in order to commit. Hence, all the advantages of total order based replication are obtained without their associated communication costs. Besides, all non-conflicting transactions do not need to be totally ordered as such protocols do. This is obtained using a dynamic priority function that guarantees the atomic commitment of transactions. The priority associated to a transaction is derived from a unique weight associated to the transaction, and the state of the transaction, which varies throughout its lifetime. This imposes the order on which conflicting transactions are applied at all nodes. Therefore, the success of a transaction that broadcasts its updates can be guessed before its submission to the underlying DBMS. Moreover, this dynamic priority function serves as a deadlock prevention function.

The rest of this paper is organized as follows: The system model is introduced in Section 2. The formalization of our protocol is presented in Section 3. Section 4 shows its correctness proof. ERP has been implemented and some experimental results as well as its comparison with other replication protocols are shown in Section 5. A brief outline of related works is given in Section 6. Finally, conclusions end the paper.

2 System Model and Definitions

For the sake of the explanation of ERP and its correctness proof, an abstraction of MADIS in a failure free environment is presented in this Section. Details about failures and the recovery process are given in [8] and the interested reader should refer to that complementary work. The system considered in this paper (Figure 1) is composed by

N nodes which communicate among them using reliable multicast [5]. We assume a fully replicated system. An application submits transactions for its execution over its local DBMS via the middleware. The replication protocol coordinates the execution of transactions among different nodes to ensure 1CS [1]. Actions in Figure 1 are shown with arrows, they describe how components interact with each other. Actions may easily be ported to the particular GCS primitives and DBMS operations.

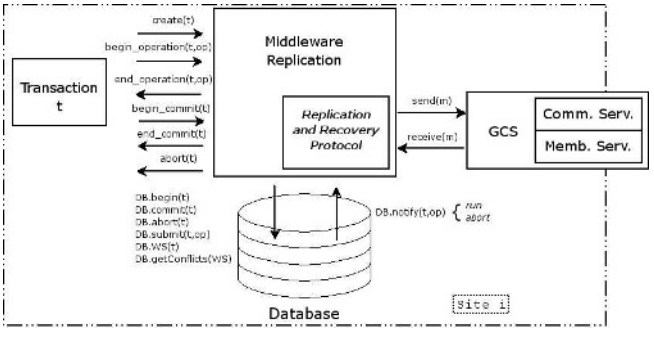


Fig. 1. Main components of the system

Database. It is assumed a DBMS ensuring ACID properties of transactions and satisfying the serializable transaction isolation level. After a SQL statement submission (denoted op) in the context of a transaction t, the $DB.notify(t, op)$ informs about the successful completion of an operation (run); or, its rollback ($abort$) due to DBMS internals. It is assumed that a transaction will only be unilaterally aborted if it is involved in a local deadlock. We also assume that after the successful completion of a submitted operation, a transaction may commit at any time. We have added two functions which are not provided by DBMSs, but may be built by standard database functions [4]. $DB.WS(t)$ retrieves the set of objects written by t and the respective SQL update statements. In the same way, the set of conflicting transactions between a write set and current active transactions is given by $getConflicts(WS(t)) = \{t' \in T\colon (WS(t') \cup RS(t')) \cap WS(t) \neq \emptyset\}$, where T is the set of system active transactions.

Transactions. Each transaction t has an identifier including the information about its *transaction master node* ($node(t)$), in order to know if it is a local or a remote transaction. It also contains information so as to obtain the weight associated to it ($weight(t)$). This value is based on its own information, such as: number of restarts, size of readset, size of writeset, node identifier and so on. All these parameters are defined by ERP at the system startup with different influence in the final weight, although ensuring its uniqueness. A transaction t created at node i ($node(t) = i$) is locally executed and starts the interaction with the rest of nodes when the application wishes to commit the transaction with the execution of *remote transactions*. Finally, ERP reports on the transaction fate to the application. For simplicity, we do not consider an application abort.

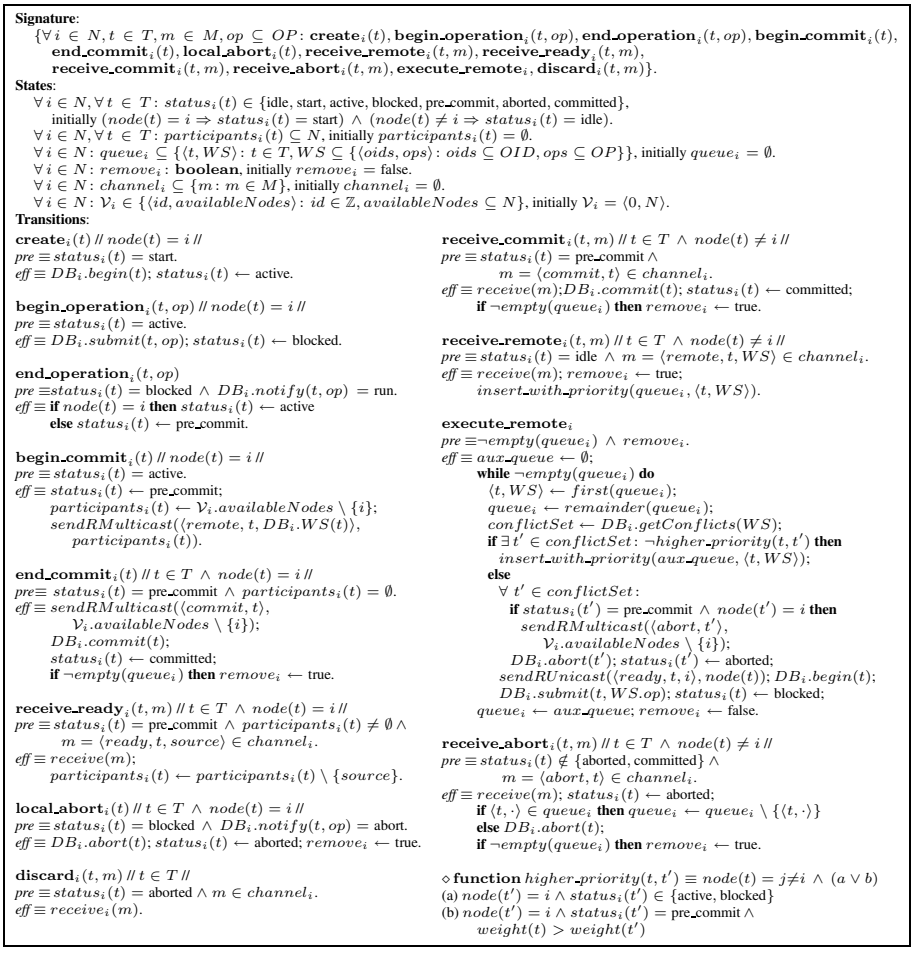


Fig. 2. State transition system for the Enhanced Replication Protocol

3 ERP Description

In this Section we use a state transition system [9] for describing ERP (introduced in Figure 2). It includes a set of state variables and actions, each one of them subscripted with the node identifier where they are considered. State variables include their domains and an initial value. Each action in the state transition system has an enabling condition (precondition, *pre* in Figure 2), a predicate over the state variables. An action is enabled if its predicate is evaluated to true on the current state. The effects of an action (*eff* in Figure 2) is a sequential program that atomically modifies the state variables; hence, new actions may become enabled while others disabled. Weak fairness is assumed for actions, i.e. if an action is continuously enabled then it will be eventually executed.

Although the state transition system seems a static structure, it defines the algorithm's execution flow. We explain such algorithm on the sequel.

A transaction t starts the execution at its master node since $status_i(t) =$ start as $node(t) = i$. It invokes the $create_i(t)$ action followed by a sequence of pairs of the form $begin\_operation_i(t, op)$ and $end\_operation_i(t, op)$. The $begin\_operation_i(t, op)$ invocation submits the SQL statement to the database ($DB_i.submit(t, op)$) and $status_i(t) =$ blocked. The transaction may be aborted due to a local deadlock resolution by the DBMS replica, as long as $status_i(t) =$ blocked, or an ERP decision (that will be discussed afterwards). The $end\_operation_i(t, op)$ action will be eventually invoked after the operation is completed in the database and the local transaction may submit a new statement. Once the transaction is done it requests its commitment, by means of the $begin\_commit_i(t)$ action, as $status_i(t) =$ active. This action initializes the variable $participants_i(t)$ to the set of reachable nodes excluding itself. ERP starts to work. Until now only the underlying DBMS was managing the concurrency control. ERP collects the writeset of the transaction and multicasts a $remote$ message to the rest of available nodes and changes its $status_i(t)$ to pre_commit. This emphasizes that it is a local transaction that has propagated its updates to the rest of available nodes.

ERP includes a $queue$ so that each time a transaction t is delivered at a node via a $remote$ message (the $receive\_remote_j(t, \langle remote, t, WS\rangle)$ action, with $j \neq i \in N$), it is firstly enqueued (arranged by $weight(t)$) and the $remove_j$ state variable is set to true. This last variable governs the time when the $queue_j$ has to be inspected, i.e. when the $execute\_remote_j$ action is called. The straightforward points of checking $queue_j$ are: when database resources are released, such as a transaction commit or rollback, and when a new $remote$ message arrives, see Figure 2. The execution of $execute\_remote_j$ disables $remove_j$ and iterates through all the transactions contained in $queue_j$ in order to check if any of these transactions has more priority than any other conflicting transaction currently submitted to the database, this is done to prevent distributed deadlock cycles between nodes. If so, the delivered transaction will send the $ready$ message to the master node and all updates will be executed in the context of another local transaction, called *remote transaction*. Before executing this remote transaction all local conflicting transactions must be firstly aborted, as this transaction has sent the $ready$ message, it may not be involved in any local deadlock. Otherwise, when the enqueued transaction has not enough priority, it will remain in $queue_j$ until it reaches the highest priority or its master node decides to abort it. One can note that several transactions (that do not conflict and have enough priority) can be submitted in one execution of $execute\_remote_j$. In either case, $ready$ messages are collected at the master node, via the $receive\_ready_i(t, \langle ready, t, j\rangle)$ action. Once all of them have been received ($participants_i(t) = \emptyset$), the $end\_commit_i(t)$ is enabled and the master node commits and multicasts a $commit$ message. The $receive\_commit_j(t, \langle commit, t\rangle)$ action will not be invoked in the rest of replicas until the updates have not been done ($status_j(t) =$ pre_commit) as it can be seen in the enabling condition of this action.

The priority between transactions is defined by the $higher\_priority(t, t')$ function where t and t' play the role of an enqueued and an executing transaction respectively. Recall that ERP does not abort a *remote transaction* already submitted to the database unless its master node decides to do so. Hence, an enqueued transaction will have

more priority than those local transactions, $t'.node = i$, still executing SQL statements ($status_i(t') \in \{\text{active}, \text{blocked}\}$) or local transactions in pre_commit whose $weight(t')$ is lower than $weight(t)$. Otherwise, an enqueued transaction will remain enqueued. Therefore, the transaction master node exclusively decides the outcome of the transaction.

Hence, ERP has modified the 2PC rule of O2PL by the use of this function. It allows a *remote transaction* to send the $ready$ message to the master node before its completion in the database. Thus, the response time $\theta_{r_{O2PL}}(t)$ of a transaction t ($node(t) = i$) with O2PL in the middleware architecture is determined by the sum of the following times: the transaction processing at the master node, $\theta_{DB_i}(t)$; multicasting the $remote$ message to the rest of nodes, $\theta_{MC}(t)$; transaction updates processing at the rest of available nodes $\theta_{DB_j}(t)$, with $j \in N \setminus \{i\}$; and, finally, each remote node sending the $ready$ message back to the master node, $\theta_{UC_j}(t)$. Therefore, we have $\theta_{r_{O2PL}}(t) \approx \theta_{DB_i}(t) + \theta_{comm}(t) + \max_j(\theta_{DB_j}(t))$, with $\theta_{comm}(t)$ grouping all communication costs. This response time is a consequence of the 2PC origin of the O2PL, and it is limited by the slowest remote transaction execution, since it waits for applying the updates at all nodes before committing. Thus, if we send the $ready$ message back once the transaction has overcome the deadlock prevention function and before it has been submitted to the database, we reduce the transaction response time to the following: $\theta_{r_{ERP}}(t) \approx \theta_{DB_i}(t) + \theta_{comm}(t)$. This time is decreased because it does not need to wait for the execution of the remote transactions, therefore we get rid of $\max_j(\theta_{DB_j}(t))$.

A *local transaction* is the single kind of transaction that may be aborted by the DBMS while it is executing SQL statements. Hence, $local\_abort_i(t, op)$ may be invoked if the DBMS may not execute the op statement contained in the $begin\_operation_i(t, op)$ due to an internal deadlock resolution; thus, $DB_i.notify(t, op) = \text{abort}$. It is important to note that ERP aborts all local conflicting transactions before the execution of a *remote transaction*; hence, it may never be involved in a local deadlock at the time it is submitted to the database. An aborted local transaction may be in the pre_commit state, in that case it will multicast an $abort$ message that will enable the $receive\_abort_j(t, \langle abort, t\rangle)$ action. In order to simplify the algorithm's presentation, we assume that the writeset of a remote transaction is atomically executed in order to avoid its concurrent execution with local transaction operations that may lead to an abortion of the former. Therefore a remote transaction may only be aborted by its master node.

4 Correctness Proof

This Section contains the most important proofs (atomicity and 1CS) of ERP in a failure free environment. For a more detailed description of the correctness proof the reader is referred to [8]. We continue using the notation and definitions of a state transition system [9]. For each ERP's action π, the enabling condition defines a set of state transitions, that is: $\{(p, \pi, q)$, p, q are states; π is an action; p satisfies $pre(\pi)$; and q is the result of executing $\mathit{eff}(\pi)$ in $p\}$. An execution, α, is a sequence of the form $s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ where s_z is a state, π_z is an action and every (s_{z-1}, π_z, s_z) is a transition of π_z. An execution is finite if it always finishes in a state, or infinite if not. Every finite prefix of an infinite execution is a finite execution. A state is reachable if it is the end of a finite execution. All possible finite executions are sufficient for defining safety proper-

ties. Liveness properties require the notion of fair execution. We assume that each ERP action requires weak fairness.

We firstly start proving that ERP is deadlock free. Local deadlocks are handled by the $local\_abort_i(t, op)$ action. Remote transactions are only involved in distributed deadlocks. The use of priorities avoids distributed deadlocks in the system. A transaction t_i ($node(t_i) = i$) *waits for* another transaction t_j ($node(t_j) = j$) if and only if t_i is remote at node j such that $weight(t_i) < weight(t_j)$, thus $t_i \in queue_j$. Assume there exist a cycle in the system ($t_0 \rightarrow t_1 \rightarrow \cdots \rightarrow t_{N-1} \rightarrow t_0$). In such a case $t_{i \bmod N}$ waits for $t_{(i+1) \bmod N}$ for i: $0 \ldots N-1$. Hence, we have that $weight(t_0) < weight(t_1) \cdots < weight(t_{N-2}) < weight(t_{N-1})$ and, if we continue with the cycle, $weight(t_{N-1}) < weight(t_0)$ which is a contradiction. Thus, the system is deadlock free.

The next Property formalizes the $status$ transition for a given transaction $t \in T$ in ERP. It points out that some $status$ transitions are unreachable, i.e., if $s_k.status_j(t) =$ pre_commit and $s_{k'}.status_j(t) =$ committed with $k' > k$. There is no action in α such that $s_{k''}.status_j(t) =$ aborted with $k' > k'' > k$, as it can be deduced from Figure 2.

Property 1. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be an arbitrary execution of the ERP automaton and $t \in T$. Let $\beta = s_0.status_j(t)\, s_1.status_j(t) \ldots s_{z'}.status_j(t)$ be the sequence of $status$ values of t at node $j \in N$, obtained from α by removing the consecutive repetitions of the same $status_j(t)$ value and maintaining the same order apparition in α. The following Property holds:*

1. *If $node(t) = j$ then β is a prefix of the regular expression:*
   $start \cdot active \cdot (blocked \cdot active)^* \cdot pre\_commit \cdot committed$
   $start \cdot active \cdot (blocked \cdot active)^* \cdot pre\_commit \cdot aborted$
   $start \cdot active \cdot (blocked \cdot active)^* \cdot aborted$
   $start \cdot (active \cdot blocked)^+ \cdot aborted$
2. *If $node(t) \neq j$ then β is a prefix of the regular expression:*
   $idle$ $\qquad$ $idle \cdot blocked \cdot pre\_commit \cdot committed$
   $idle \cdot blocked \cdot aborted$ $\qquad$ $idle \cdot blocked \cdot pre\_commit \cdot aborted$
   $idle \cdot aborted$

This Property is simply proved by induction over the length of α following the preconditions and effects of the ERP actions. A $status$ transition for a transaction t in Property 1 is associated with an operation on the DB module where the transaction was created, i.e. pre_commit to $committed$ involves the $DB.commit(t)$ operation. These aspects are straightforward from the inspection of Figure 2.

The following Property is needed to prove the atomicity of a transaction; that is, the transaction is either committed, or aborted, at all available nodes. It states the invariant properties of ERP. If a transaction t with $node(t) = i$ is committed at $j \neq i$, it is because it was already committed at its master node. A remote transaction currently being executed at its DB_j module ($status_j(t) =$ blocked) may only change its $status$ if its execution is completed or by an $abort$ message coming from its master node. In other words, it will never be aborted by the DB_j module. In the same way, a remote transaction in the pre_commit state may only change its $status$ if it receives a $commit$ or an $abort$ message from its master node. Finally, if a transaction is $committed$ at its master node then at the rest of available nodes it will be either $committed$ (it has

already received the $commit$ message), pre_commit (it is waiting to receive the $commit$ message) or $blocked$ (it is still applying the updates at that node).

Property 2. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be an arbitrary execution of the ERP automaton and $t \in T$, with $node(t) = i$.*

1. *If $\exists\, j \in N \setminus \{i\}$: $s_z.status_j(t) =$ committed then $s_z.status_i(t) =$ committed.*
2. *If $\exists\, z' < z$: $s_{z'}.status_j(t) = s_z.status_j(t) =$ blocked for any $j \in N \setminus \{i\}$ then $\forall\, z''$: $z' < z'' \leq z$: $\pi_{z''} \notin \{receive\_abort_j(t, \langle abort, t\rangle), end\_operation_j(t, WS.op)\}$.*
3. *If $\exists\, z' < z$: $s_{z'}.status_j(t) = s_z.status_j(t) =$ pre_commit for any $j \in N \setminus \{i\}$ then $\forall\, z''$: $z' < z'' \leq z$: $\pi_{z''} \notin \{receive\_commit_j(t, \langle commit, t\rangle), receive\_abort_j(t, \langle abort, t\rangle)\}$.*
4. *If $s_z.status_i(t) =$ committed then $\forall\, j \in N$: $s_z.status_j(t) \in \{blocked, pre\_commit, committed\}$.*

The following Lemma –liveness property– states the atomicity of committed transactions.

Lemma 1. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $t \in T$ with $node(t) = i$. If $\exists\, j \in N$: $s_z.status_j(t) =$ committed then $\exists\, z' > z$: $s_{z'}.status_j(t) =$ committed for all $j \in N$.*

Proof. If $j \neq i$ by Property 2.1 (or with $j = i$) $s_z.status_i(t) =$ committed. By Property 2.4, $\forall\, j \in N \setminus \{i\}$: $s_z.status_j(t) \in \{$blocked, pre_commit, committed$\}$. Without loss of generality, assume that s_z is the first state where $s_z.status_i(t) =$ committed and $s_z.status_j(t) =$ pre_commit (if $s_z.status_j(t) =$ blocked it is because of its submission to the DB_j module, due to $execute\_remote_j$ for t). By weak fairness of action execution, the $end\_operation_j(t, WS.op)$ action will be eventually invoked and $s_z.status_j(t) =$ pre_commit). By the effects of $\pi_z = end\_commit_i(t)$, we have that $\langle commit, t\rangle \in s_z.channel_j$. By Property 2.4 invariance either $s_z.status_j(t) =$ committed or $s_z.status_j(t) =$ pre_commit and $\langle commit, t\rangle \in s_z.channel_j$. In the latter case the $receive\_commit_j(t, \langle commit, t\rangle)$ action is enabled. By weak fairness assumption, the action will be eventually executed, thus $\exists\, z' > z$: $\pi_{z'} = receive\_commit_j(t, \langle commit, t\rangle)$. Thus, by its effects, $s_{z'}.status_j(t) =$ committed.

In a similar way, when a transaction is aborted, it is aborted at all nodes, as stated in the following Lemma. The proof is very simple, by inspection of the ERP actions.

Lemma 2. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $t \in T$ with $node(t) = i$. If $s_z.status_i(t) =$ aborted then $\exists\, z' \geq z$: $s_{z'}.status_j(t) =$ idle for all $j \in N \setminus \{i\}$ or $s_{z'}.status_j(t) =$ aborted for all $j \in N$.*

Before continuing with the correctness proof we have to add a definition dealing with causality between actions. Some set of actions may only be viewed as causally related to another action in any execution α. We denote this fact by $\pi \prec_\alpha \pi'$. For example, assuming t is a committed transaction with $node(t) = i \neq j$, the following *happens-before* relation $begin\_commit_i(t) \prec_\alpha receive\_remote_j(t, \langle remote, t, WS\rangle)$ is held, see Figure 2. This is clearly seen by the effects of the $begin\_commit_i(t)$ action: it sends a $\langle remote, t, DB_i.WS(t)\rangle$ to all $j \in N \setminus \{i\}$. This message will be eventually received by

j that enables the $receive\_remote_j(t, \langle remote, t, WS\rangle)$ action. As $status_j(t) =$ idle, and by weak fairness of actions, it will be eventually executed. However, this fact is delegated to the $execute\_remote_j$ action. The following Lemma indicates that a transaction is $committed$ if it has received every $ready$ message from all available remote replicas.

Lemma 3. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $t \in T$ be a committed transaction, $node(t) = i$, then the following happens-before relations hold: $\forall j \in N \setminus \{i\}$: $begin\_commit_i(t) \prec_\alpha receive\_remote_j(t, \langle remote, t, WS\rangle) \prec_\alpha execute\_remote_j(t) \prec_\alpha receive\_ready_i(t, \langle ready, j\rangle) \prec_\alpha end\_commit_i(t) \prec_\alpha receive\_commit_j(t, \langle commit, t\rangle)$.*

The following Lemma emphasizes the *happens-before* relation for remote transactions. It is based on Property 1.2 which establishes the relation between $status$ transitions for remote transactions to their respective algorithm actions. This will serve in order to set up the relation for a transaction t, with $node(t) = i \neq j$, between the $execute\_remote_j$, that submits t to the DB_j module, and the pair $end\_operation_j(t, WS.op)$ and $receive\_commit_j(t, \langle commit, t\rangle)$ actions. This is needed due to the fact that with Lemma 3 there is no point where this causal relation may be put in.

Lemma 4. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $t \in T$ be a committed transaction, $node(t) = i$, then the following happens-before relations hold: $\forall j \in N \setminus \{i\}$: $receive\_remote_j(t, \langle remote, t, WS\rangle) \prec_\alpha execute\_remote_j(t) \prec_\alpha end\_operation_j(t, WS.op) \prec_\alpha receive\_commit_j(t, \langle commit, t\rangle)$.*

In order to define the correctness of our replication protocol we have to study the global history (H) of committed transactions ($C(H)$) [1]. We may easily adapt this concept to ERP. Therefore, a new auxiliary state variable, H_i, is defined in order to keep track of all the DB_i operations performed on the local DBMS at node i. For a given α execution of ERP, $H_i(\alpha)$ plays a similar role as the local history at node i, H_i, as introduced in [1] for the DBMS. In the following, only committed transactions are part of the history, deleting all operations that do not belong to transactions committed in $H_i(\alpha)$. The serialization graph for $H_i(\alpha)$, $SG(H_i(\alpha))$, is defined as in [1]. An arc and a path in $SG(H_i(\alpha))$ are denoted as $t \rightarrow t'$ and $t \xrightarrow{*} t'$ respectively. Recall that our local DBMS produces serializable histories; thus, $SG(H_i(\alpha))$ is acyclic and the history is strict. Thus, for any execution resulting in local histories $H_1(\alpha), H_2(\alpha), \ldots, H_N(\alpha)$ at all nodes its serialization graph, $\cup_k SG(H_k(\alpha))$, must be acyclic so that conflicting transactions are equally ordered in all local histories. Recall that the correctness criterion is 1CS.

Before showing the correctness proof, we need an additional Property relating the transaction isolation level of the underlying DB modules to the automaton execution event ordering. The following Property and Corollary establish a property about local executions of committed transactions and their respective ERP actions.

Property 3. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be an arbitrary execution of the ERP automaton and $i \in N$. If there exist two transactions $t, t' \in T$ such that $t \xrightarrow{*} t'$ in $SG(H_i(\alpha))$ then $\exists\, z_1 < z_2 < z_3 < z_4$: $s_{z_1}.status_i(t) =$ pre_commit $\wedge$ $s_{z_2}.status_i(t) =$ committed $\wedge$ $s_{z_3}.status_i(t') =$ pre_commit $\wedge$ $s_{z_4}.status_i(t') =$ committed.*

Corollary 1. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $i \in N$. If there exist two transactions $t, t' \in T$ such that $t \xrightarrow{*} t'$ in $SG(H_i(\alpha))$ then the following happens-before relation, with the appropriate parameters, holds:*

1. $node(t) = node(t') = i$: $begin\_commit_i(t) \prec_\alpha end\_commit_i(t) \prec_\alpha begin\_commit_i(t') \prec_\alpha end\_commit_i(t')$.
2. $node(t) = i \wedge node(t') \neq i$: $begin\_commit_i(t) \prec_\alpha end\_commit_i(t) \prec_\alpha end\_operation_i(t', WS'.op) \prec_\alpha receive\_commit_i(t', \langle commit, t' \rangle)$.
3. $node(t) \neq i \wedge node(t') = i$: $end\_operation_i(t, WS.op) \prec_\alpha receive\_commit_i(t, \langle commit, t \rangle) \prec_\alpha begin\_commit_i(t') \prec_\alpha end\_commit_i(t')$.
4. $node(t) \neq i \wedge node(t') \neq i$: $end\_operation_i(t, WS.op) \prec_\alpha receive\_commit_i(t, \langle commit, t' \rangle) \prec_\alpha end\_operation_i(t', WS'.op) \prec_\alpha receive\_commit_i(t', \langle commit, t' \rangle)$.

Proof. By Property 3, $\exists\, z_1 < z_2 < z_3 < z_4$: $s_{z_1}.status_i(t) =$ pre_commit $\wedge$ $s_{z_2}.status_i(t) =$ committed $\wedge$ $s_{z_3}.status_i(t') =$ pre_commit $\wedge$ $s_{z_4}.status_i(t') =$ committed. Depending on $node(t)$ and $node(t')$ values the unique actions that modify their associated $status$ to the given values, by Property 3, are the ones indicated in the Corollary.

If we have two conflicting transactions $t, t' \in T$, with $node(t) \neq i$ and $node(t') \neq i$, such that $t \rightarrow t'$ in $SG(H_i(\alpha))$, then the $execute\_remote_i$ action that submits t' to the database must be executed after committing t, via the $receive\_commit_i(t, \langle commit, t \rangle)$ action. The next Lemma states how the *happens-before* relation affects to two committed transactions executing at a remote node.

Lemma 5. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $i \in N$. If there exist two committed transactions $t, t' \in T$ with $node(t) = j \neq i$ and $node(t') = k \neq i$ such that $t \xrightarrow{*} t'$ in $SG(H_i(\alpha))$ then the following happens-before relation hold: $\forall\, i \in N \setminus \{k, j\}$: $execute\_remote_i(t) \prec_\alpha receive\_commit_i(t, \langle commit, t \rangle) \prec_\alpha execute\_remote_i(t') \prec_\alpha receive\_commit_i(t', \langle commit, t' \rangle)$.*

The same may be applied to two conflicting transactions $t, t' \in T$ with $node(t) = i$ and $node(t') \neq i$, such that $t \rightarrow t'$ in $SG(H_i(\alpha))$. The $execute\_remote_i$ action that submits t' to the database must be executed after the commitment of t by the $end\_commit_i(t)$ action. The next Lemma states how the *happens-before* relation affects to a committed transaction executing at a remote node.

Lemma 6. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton and $i \in N$. If there exist two transactions $t, t' \in T$ with $node(t) = i$ and $node(t') \neq i$ such that $t \xrightarrow{*} t'$ in $SG(H_i(\alpha))$ then the following happens-before relation hold: $\forall\, i \in N$: $begin\_commit_i(t) \prec_\alpha end\_commit_i(t) \prec_\alpha execute\_remote_i(t') \prec_\alpha end\_operation_i(t', WS'.op) \prec_\alpha receive\_commit_i(t', \langle commit, t' \rangle)$.*

In the following, we prove that the ERP protocol provides 1CS [1].

Theorem 1. *Let $\alpha = s_0\pi_1 s_1 \ldots \pi_z s_z \ldots$ be a fair execution of the ERP automaton. The graph $\cup_{k \in N} SG(H_k(\alpha))$ is acyclic.*

Proof. By contradiction. Assume there exists a cycle in $\cup_{k \in N} SG(H_k(\alpha))$. There are at least two different transactions $t, t' \in T$ and two different nodes $x, y \in N$, $x \neq y$, such

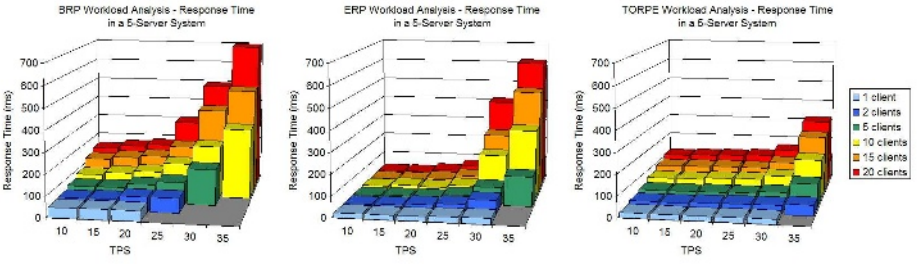


Fig. 3. Protocols Response Time Evaluation

that those transactions are executed in different order at x and y. Thus, we consider (a) $t \xrightarrow{*} t'$ in $SG(H_x(\alpha))$ and (b) $t' \xrightarrow{*} t$ in $SG(H_y(\alpha))$; being $node(t) = i$ and $node(t') = j$. There are four cases under study:

(I) $i = j = x$.
(II) $i = x \wedge j = y$.
(III) $i = j \wedge i \neq x \wedge i \neq y$.
(IV) $i \neq j \wedge i \neq x \wedge i \neq y \wedge j \neq x \wedge j \neq y$.

In the following, we simplify the notation. The action names are shortened, i.e. $begin\_commit_x(t)$ by $bc_x(t)$; $end\_commit_x(t)$ by $ec_x(t)$; as each $execute\_remote_x$ action may execute a set of transactions, $K \subseteq T$, we denote it by $er_x(k)$, with $k \in K$; $receive\_ready_x(t, \langle ready, t, l\rangle)$, with $l \in N$, by $rr_x(t, l)$; $end\_operation_x(t, op)$ by $eo_x(t)$; and, $receive\_commit_x(t, \langle commit, t\rangle)$ by $rc_x(t)$.

(I). By Corollary 1.1 for (a): $bc_x(t) \prec_\alpha ec_x(t) \prec_\alpha bc_x(t') \prec_\alpha ec_x(t')$. ($i$)
By Corollary 1.4 for (b): $eo_y(t') \prec_\alpha rc_y(t') \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. Applying Lemmas 4 and 5 for t and t': $er_y(t') \prec_\alpha eo_y(t') \prec_\alpha rc_y(t') \prec_\alpha er_y(t) \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. ($ii$)

For (i), via Lemma 3 for t, we have the following: $bc_x(t) \prec_\alpha er_y(t) \prec_\alpha rr_x(t, y) \prec_\alpha ec_x(t) \prec_\alpha bc_x(t') \prec_\alpha ec_x(t')$. Taking into account Lemma 3 for t' and Lemma 5 for t and t': $bc_x(t) \prec_\alpha er_y(t) \prec_\alpha rr_x(t, y) \prec_\alpha ec_x(t) \prec_\alpha bc_x(t') \prec_\alpha er_y(t') \prec_\alpha rr_x(t', y) \prec_\alpha ec_x(t') \prec_\alpha rc_y(t')$. Therefore, we have that $er_y(t) \prec_\alpha rc_y(t')$ in contradiction with (ii).

(II). By Corollary 1.2 for (a): $bc_x(t) \prec_\alpha ec_x(t) \prec_\alpha eo_x(t') \prec_\alpha rc_x(t')$. By Lemma 6 for t and t': $bc_x(t) \prec_\alpha ec_x(t) \prec_\alpha er_x(t') \prec_\alpha rc_x(t')$. ($i$)
By Corollary 1.2 for (b): $bc_y(t') \prec_\alpha ec_y(t') \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. Applying Lemma 6 for t' and t: $bc_y(t') \prec_\alpha ec_y(t') \prec_\alpha er_y(t) \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. ($ii$)

By Lemma 3 for t: $bc_x(t) \prec_\alpha er_y(t) \prec_\alpha rr_x(t, y) \prec_\alpha ec_x(t)$, via ($i$), $\prec_\alpha er_x(t') \prec_\alpha rr_y(t', x) \prec_\alpha ec_y(t') \prec_\alpha rc_x(t')$. Thus $er_y(t) \prec_\alpha ec_y(t')$ in contradiction with (ii).

(III). As x and y are different nodes from the transaction master node, only one of them will be executed in the same order as in the master node. If we consider the different one with the master node we will be under assumptions considered in CASE (I).

(IV) By Corollary 1.4 for (a): $eo_x(t) \prec_\alpha rc_x(t) \prec_\alpha eo_x(t') \prec_\alpha rc_x(t')$. Applying Lemmas 4 and 5 for t and t' at x: $er_x(t) \prec_\alpha eo_x(t) \prec_\alpha rc_x(t) \prec_\alpha er_x(t') \prec_\alpha eo_x(t') \prec_\alpha rc_x(t')$. ($i$)

By Corollary 1.4 for (b): $eo_y(t') \prec_\alpha rc_y(t') \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. If we apply Lemmas 4 and 5 for t' and t at y: $er_y(t') \prec_\alpha eo_y(t') \prec_\alpha rc_y(t') \prec_\alpha er_y(t) \prec_\alpha eo_y(t) \prec_\alpha rc_y(t)$. ($ii$)

By Lemma 3 for t at x and y: $bc_i(t) \prec_\alpha er_y(t) \prec_\alpha rr_i(t,y) \prec_\alpha ec_i(t) \prec_\alpha rc_x(t)$. Via Corollary 1.4 for (a): $bc_i(t) \prec_\alpha er_y(t) \prec_\alpha rr_i(t,y) \prec_\alpha ec_i(t) \prec_\alpha rc_x(t) \prec_\alpha er_x(t') \prec_\alpha rr_j(t',x) \prec_\alpha ec_j(t') \prec_\alpha rc_y(t')$. Therefore, we have that $er_y(t) \prec_\alpha rc_y(t')$ in contradiction with (ii).

5 Experimental Results

We have implemented the ERP protocol in the MADIS architecture, in order to test the performance improvement of the ERP protocol against other approaches, such as BRP (based on a termination voting similar to a 2PC) and TORPE (based on total order delivery of write-sets, removing the voting phase). For all the experiments, we used a cluster of 5 workstations (Pentium IV 2.8GHz, 1GB main memory, 80GB IDE disk) connected by a full duplex Fast Ethernet network. We have implemented an *ad hoc* reliable multicast using TCP for BRP and ERP, whilst Spread 3.17 was in charge of the total order multicast needed by TORPE. PostgreSQL 7.4 was used as the underlying DBMS. The database consists of 30 tables each containing 1000 tuples. Each table has the same schema: two integers, one being the primary key. Transactions consist of a number of update operations each one modifying a given tuple randomly chosen from a table of the database. The interarrival time between the submission of two consecutive transactions is uniformly distributed. The workload is denoted by the number of transactions submitted per second (TPS). All tests were run until 2000 transactions were executed.

These experiments test how the three replication protocols cope with increasing number of users and workloads. Workload was increased steadily from 10 to 35 TPS. For each workload, several tests were executed varying the number of clients from 1 to 20 in the whole system. Figure 3 shows the response time obtained in this experiment for the three presented protocols. As a general rule, the maximum throughput is limited since a client can only submit one transaction at a time, and hence, the submission rate per client is limited by the response time. With one client TORPE and ERP's response times are below 25 and 28 ms respectively, but BRP ones are close to 50 ms and it is not possible to achieve the desired throughput. The abortion rates for this experiment are low, between 0% and 2%, due to the random nature of the generated numbers.

Results revealed that BRP presents the worst behavior of the presented protocols, due to its 2PC transaction termination. In ERP, the remote nodes send the $ready$ message once conflicts and priority rules are checked before executing the updates. In TORPE, the master node does not wait for any response from remote nodes, only waits for the delivery of the messages in total order. As seen in Figure 3, ERP response times keep between 20 and 70 ms for a given number of clients with workloads up to 25 TPS. TORPE response times, working with intermediate loads, remain between 20 ms with 1 client and 125 ms with 20 clients.

Increasing the workload and the multiprogramming level only results in higher response times, due to the administration overhead (process switches, communication to/from the client) and contention at specific resources. As shown in Figure 3, BRP and ERP are more affected by this overhead than TORPE is. This happens since we have implemented the ad hoc reliable multicast inside the ERP and BRP protocols, hence they support the load of the protocol itself and the communication between nodes. On the

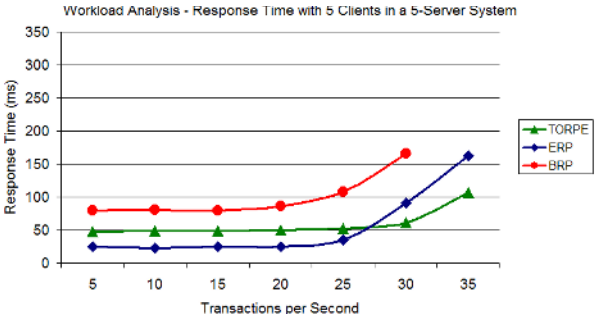


Fig. 4. Response time of the replication protocols varying the submission rate

other hand, TORPE uses an underlying GCS to manage communication among nodes that runs independently of the protocol. Thus, TORPE response times do not increase so much as in the other protocols at high workloads. That is the reason why there is a crossing point between ERP and TORPE in Figure 4, where only the configuration with 5 clients is analyzed.

6 Related Works

There are a lot of replication protocols defined in the literature, modifying the DBMS core [1,2,3]. However, we will highlight those developed for middleware solutions. For instance, with optimistic atomic broadcast, as described in [3], messages are delivered as they are received, making possible a fast remote writeset application, although waiting for the final ordered message delivery in order to commit the transaction. Thus, only those remote transactions whose writeset did not follow the total order are rolled back, reapplying them in the correct order. This idea has been applied in [10] for WAN networks in the GlobData project. Respectively, a more aggressive version of the optimistic atomic broadcast [3] in a middleware architecture is presented in [4]. Database stored procedures are executed as transactions defining a conflict class. Transactions issued by users are delivered using the optimistic atomic broadcast to all nodes but the outcome of transactions is only decided at the master node of the respective conflict class. Hence, remote nodes do not even have to wait for the definitive total order to execute a transaction. It additionally provides good scalability results. The BRP and ERP may accept any SQL statement, hence they are more flexible; however, ours present a higher overhead since they propagate the SQL statements and we do not try to balance the workload.

7 Conclusions

In this paper, we present a middleware database replication protocol, called ERP. It is an adaptation of O2PL where the 2PC rule has been modified in order to improve O2PL

performance. We have proved that ERP is 1CS –and this has been, up to our knowledge, the first prove of this kind for any O2PL-based replication protocol–, given that the underlying DBMSs feature a serializable transaction isolation level. This replication protocol has the advantage that no specific DBMS tasks have to be re-implemented. The underlying DBMS performs its own concurrency control and the replication protocol complements this task with replica control.

ERP is an eager update everywhere replication protocol. All transaction operations are firstly performed on its master node, and then all updates are grouped and sent to the rest of nodes using a reliable multicast. However, our algorithm is liable to suffer distributed deadlock. Hence, we have defined a deadlock prevention schema that orders transactions, which is based on the transaction state and associated weight. Besides, as it totally orders transactions, ERP will know if a transaction may proceed or not before its submission to the DBMS. This allows us to get rid of the of the strict 2PC rule.

ERP has been implemented in MADIS, as well as an adaptation of O2PL (BRP) and a total order based (TORPE) protocols. They have been compared against each other with a specific benchmark. Results highlight the benefits of ERP when compared to BRP. However, TORPE shows better performance in heavy loaded environments, due to our reliable multicast implementation, whilst ERP is the best in the rest of tests, due to the overhead introduced by Spread to provide total order. Thus, ERP may be considered as an intermediate solution between 2PC and total order based protocols.

References

1. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison Wesley (1987)
2. Carey, M.J., Livny, M.: Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst. **16** (1991) 703–746
3. Kemme, B., Pedone, F., Alonso, G., Schiper, A., Wiesmann, M.: Using optimistic atomic broadcast in transaction processing systems. Trans. Knowl. Data Eng. **15** (2003) 1018–1032
4. Jiménez-Peris, R., Patiño-Martínez, M., Kemme, B., Alonso, G.: Improving the scalability of fault-tolerant database clusters. In: ICDCS. (2002) 477–484
5. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Comput. Surv. **33** (2001) 427–469
6. Défago, X., Schiper, A., Urbán, P.: Total order broadcast and multicast algorithms: Taxonomy and survey. ACM Comput. Surv. **36** (2004) 372–421
7. Irún, L., Decker, H., de Juan, R., Castro, F., Armendáriz, J.E., Muñoz, F.D.: MADIS: A slim middleware for database replication. Springer LNCS **3648** (2005) 349–359
8. Armendáriz, J.E.: Design and Implementation of Database Replication Protocols in the MADIS Architecture. PhD thesis, Universidad Pública de Navarra, Pamplona, Spain (2006)
9. Shankar, A.U.: An introduction to assertional reasoning for concurrent systems. ACM Comput. Surv. **25** (1993) 225–262
10. Rodrigues, L., Miranda, H., Almeida, R., Martins, J., Vicente, P.: The GlobData fault-tolerant replicated distributed object database. Springer LNCS **2510** (2002) 426–433

Reconfiguration of Information Management Framework Based on Adaptive Grid Computing

Eun-Ha Song[1], Sung-Kook Han[1], Laurence T. Yang[2], and Young-Sik Jeong[1,*]

[1] Department of Computer Engineering, Wonkwang University
344-2 Shinyong-Dong, Iksan, 570-749, Korea
{ehsong, skhan, ysjeong}@wku.ac.kr
[2] Department of Computer Science, St. Francis Xavier University
Antigonish, NS, B2G 2W5, Canada
lyang@stfx.ca

Abstract. In this paper, GridIMF provides the users with consistency while adapting to variability grid information. GridIMF designs hierarchical 3-tier information management model in accordance with participating intention, roles and operating strategies of grid information. In order to manage grid information in effective ways, GridIMF provides grid service while dividing to GVMS and GRMS. GVMS suggests optimal virtual organization selection mechanism for improving performance of specific applications and LRM auto-recovery strategy that treat faults of virtual organization. GRMS supports adaptive performance-based task allocation method for load balancing and fault tolerance. State monitoring and visualization view provides adaptability of managing grid information. Application proxy removes inter-dependency between service objects of GridIMF and application objects. For analyzing GridIMF executability, it adapts two fractal image processing those characteristics are different.

1 Introduction

Grid resources are the key elements to support performance of grid application. If computers are the grid resources, it can be specifications of hardware and software in a limited definition or it can be a certain organization for specialized goals in a broad definition. Grid resources potentially have policies about another fields, and they are controlled by another institutions. Therefore, grid computing environment requires reconfiguration in order to support utilizing diverse resources all over the world, and achieve the objects of computational grid. In grid computing environment, it is needed to design a framework that can control upper requirements including representation of grid resources and control of grid operation [1][2][3][4].

In this paper, GridIMF(Grid Information Management Framework) is established with accepting characteristics of grid resources and supporting grid services at the same time. In this point, grid resources mean a tool to perform an operation. Meanwhile, metadata include computing information (CPU, memory and CPU-Load) and networking information (bandwidth, wait time). This paper defines virtual

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 538 – 548, 2006.
© Springer-Verlag Berlin Heidelberg 2006

organization, resources, and metadata as grid resources. That is, grid resources have integral meaning regardless of scale of information or its characteristics [5][6].

According to the role of grid information, GridIMF designs hierarchical 3-tier information management model and it designs 3-layer framework model in accordance with managing of grid information. A framework model provides grid services with dividing GVMS(Grid Virtual Organization Management System) for managing virtual organization and GRMS(Grid Resource Management System) for managing metadata. This framework shows adaptability and validity of supported service with two differential fractal image processing.

2 Related Works

As the grid logically couples multiple resources owned by different individuals or organizations, the choice of the right model for resource management architecture plays a major role in its eventual success. There are models of grid resource management such as Hierarchical, Abstract Owner, and Economy Model, and Table 1 shows a representative system that follows aforementioned models [7].

Table 1. Three Models for a Grid Resource Management Architecture

Model	System
Hierarchical Model	Globus [8], Legion, Ninf, NetSolve, GridIMF
Abstract Owner Model	-
Economy/Market Model	Nimrod/G, JaWS, Myriposa, JavaMarket

GridIMF designed by this paper applies Hierarchical Model that a lot of current grid systems follow. GridIMF includes passive elements of Hierarchical Model such as resources, tasks, jobs, schedules, and GRMS suggests active elements of Hierarchical Model such as scheduler, information service, and domain control agent. GridIMF admits managing functions about information service and resource management that the representative model for Hierarchical Model, Globus toolkit, includes. But it is inefficient in the matter of management for metadata as it has the uniformed change and delivery cycle which ignore the characteristic of data. Globus Toolkit can cause the degradation of grid performance. GridIMF provides GVMS and GRMS supporting Globus toolkit's MDS and GRAM with considering management of metadata. GVMS applies the optimum virtual organization selective mechanism which maximizes the use of resource in the user point of view and optimizes it in the application point of view; and suggests auto-recovery strategy which confronts dynamically to the VO fault. GRMS receives the metadata information within local resource periodically through virtual observer and confronts the load balancing and local resource fault which was resulted in reducing total amount time of computing with adaptive performance-based task allocation which runs scheduler. GridIMF understands the Grid information in real-time and adds the monitoring technique to enable the effective sharing and putting practice. Monitoring technique can be adapted to the variableness of Grid information and the various forms of virtualization view have easy way of management [9]. Table 2 shows the comparison of Globus toolkit and supportive system which was presented in this system.

Table 2. Globus vs. GridIMF

System / Activity Functions	Globus	**GridIMF**	Remark
Scalability of grid	●	●	Hierarchical 3-tier information management model
Dynamic VO configuration	▲	●	
Availability for service	●	●	Optimum virtual organizations selection mechanism and auto-recovery strategy
Deletion of replicate operation at fault	×	●	
Information state monitoring	●	●	PDH Library, (LRM/RP)Virtual observer
Information state visualization	×	●	Variable visualization view
Load balancing strategy	▲	●	Performance-based Task Allocation Method
Fault tolerance strategy	▲	●	
Independency among application	●	●	Application Proxy
User access interface	●	▲	Java Applet

(●: support, ▲: a partial support, ×: nonsupport)

3 Grid Information Management Framework

3.1 GridIMF Architecture

GridIMF aims the high performance and the distributed computing. Basically in the infrastructure, the dynamic and effective maintenance, management, and usage of metadata which is the detailed element of grid information should be available. In this paper, as shown in figure 1, 3-layer framework model is constructed and the existing computing resources which are preserved in one site can be formed into the GridIMF formation.

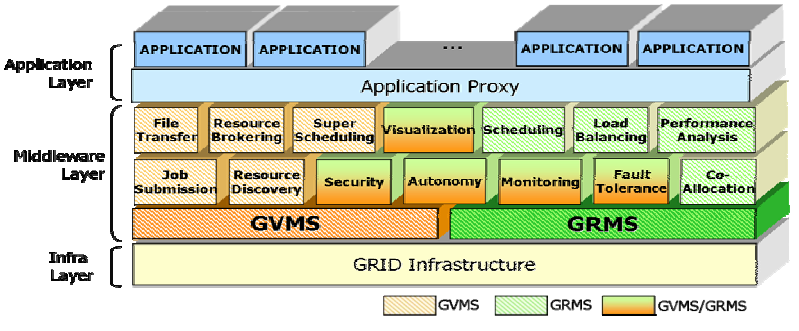


Fig. 1. 3-layer framework model

Infra layer includes the base task to connect the different characteristic constituent into the integrated resource. Middleware layer has a role to show the available information to the user into one system and adds the function which was suggested in this paper to the general middleware function. The management of metadata which was physically divided individually is divided into GRMS and the management of virtual

organization is divided into GVMS. In application layer, the grid applications which need much calculation can approach to the grid information which is connected through infra layer and application and dependent middleware layer service are provided through application proxy.

3.2 Function Model of the GridIMF Components

GridIMF divides the components of grid into RR(Resource Requester), RP(Resource Provider), LRM(Local Resource Manager) and GIM(Global Information Manager) depending on the intention and purpose of participation. Fig. 2 shows the structure of architecture layer and control stream of suggested GridIMF components.

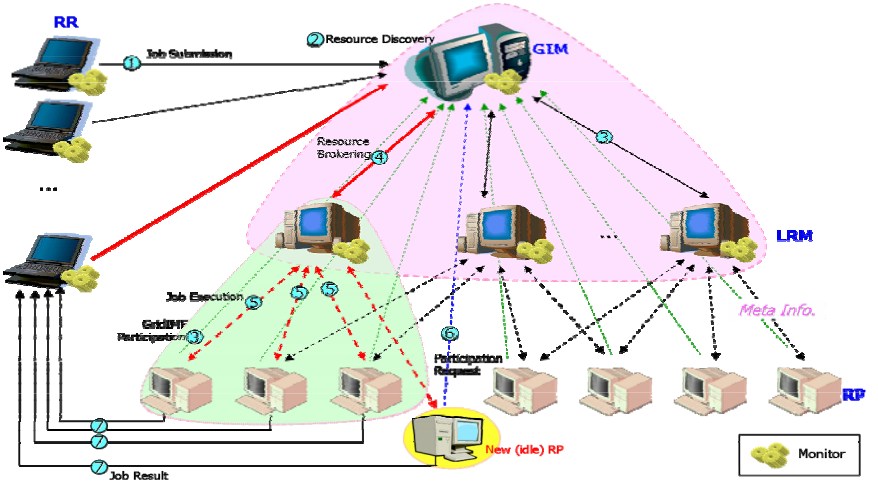


Fig. 2. Structure of layer and control stream of GridIMF components

RR is the grid user and requests job submission to GIM. RR can request several jobs at the same time, only receives the corresponding result of requested job and does not participate in computation. RR plays a role of interface delivering information of independent application. GIM is the top level manager and is a kind of resource brokering proxy connecting between resource and user who wants to use grid in remote. GIM manages the connection by providing common interface of components and executes job submission and control management as proxy. GIM is a super scheduler which brings the list of LRM satisfying based on job specification of specific application and permits the connection. LRM gets the delegation collecting resource and allocating task for requested job. LRM selects RP through resource discovery, intermediate the connection between RP and RR and makes core scheduling between remote resources. RP is a group which permits the execution for a part of task instead of huge scale of job and collects the information of metadata. RP delivers the executable resource, performs the operation and transmits the result.

4 Grid Virtual Organization Management System

4.1 Task Brokering Rule

Virtual organization has different policy, purpose and size. Grid application is finished its task by virtual organization with the scheduling. Therefore, the performance of application is the selection of virtual organization appropriate to the task. GridIMF defines 4 kinds of factors for selecting optimum virtual organization:

- Possibility of application execution: One application is assumed to be executed by one virtual organization. One LRM is possible to execute applications independently. GIM provides a list of application which is already executed and should be executed. LRM selects application which can be executed. The first stage selection depends on LRM which knows the capability and size of its virtual organization.
- Idle State: GIM senses the wait state by LRM virtual observer that is going to execute the requested task and then decides the possibility of execution.
- Application Power: The operation result of grid application is recorded as log file. The attribute of log file is the application name, execution time, LRM information, number of LRM and RP, total amount of static CPU speed, etc.
- LRM Performance Index: Performance index can be obtained by the analysis of application log file. It selects a value close to the expected value in proportion to the average speed of CPU and number of nodes for executing requested application.

4.2 LRM State Control and Auto-recovery Strategy

GIM is the manager of dynamic LRM so that transmission of control message is frequent related to LRM. GridIMF makes LRM virtual observer that gathers information of LRM and plays communication broker, and separates operation and control. LRM virtual observer is the monitor which makes scheduling the task and manages RPs inside virtual organization. LRM auto-recovery strategy senses the LRM fault and secures the availability of service by obtaining the corresponding resource from another LRM. RR can be possible to use the grid information and to change the specification of executing application until the connection of GridIMF is disconnected. The fault information of LRM is accomplished by LRM virtual observer. When LRM fault is found, it looks for a new LRM in idle state. RR sends address, remote object reference and task table for the reference and operation is automatically executed by a new LRM. When the search for LRM is failed, RR is in wait state and added into the waiting list. The management of RR task table and remote object reference do not have replicated operation of LRM which generated fault.

4.3 GVMS Monitoring Information Visualization

GVMS provides the virtualization view of various forms to manager and system planner through real-time monitor. Monitoring and virtualization data is limited to LRM and RP in the GIM side of hierarchical information management model and divided into Virtual Organization Monitor. It does not include metadata information of RP. The target of monitoring is RR, LRM and RP which are performing and holding calculation. Visualization factor recorded the relative group which was traced by

virtual observer in LRM Entry and RR Entry and transferred the revised information to the view. GVMS monitoring information visualization is provided with button from GIM main frame with one frame and two views as shown in Fig. 3.

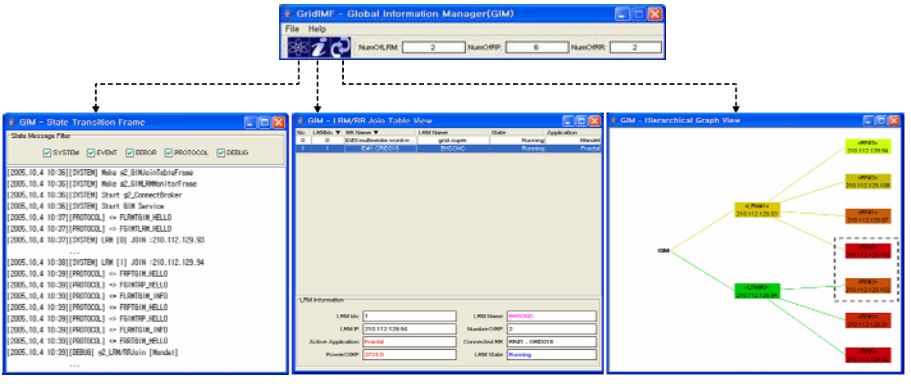


Fig. 3. Information Visualization View of GVMS

GIM state transition frame is a demon frame which runs during main frame loading. LRM/RR Join View provides the relationship between RR and LRM with table component, and the detailed information of selected line is output into text component. LRM/RP Hierarchical View provides the node structure of GridIMF with hierarchically. Fig. 3 is composed with two virtual organizations and seven RPs, but the dotted area performs RP as well as two applications.

5 Grid Resource Management System

5.1 Task Allocation Algorithms

GRMS makes the counter measure and suggests DPTA(Dynamic Performance-based Task Allocation) and APTA(Adaptive Performance-based Task Allocation) for minimizing the cost of resource. Table 3 is the suggested task allocation method.

Table 3. Task Allocation Method of GridIMF

Task Allocation Method	Performance Evaluation & Reallocation Factor	Remark
Dynamic Performance-based Task Allocation	▪ CPU Speed ▪ Job History	▪ Fault Detection ▪ Append Processor ▪ State Monitoring
Adaptive Performance-based Task Allocation	▪ CPU Speed ▪ CPU Usage ▪ Proportional Factor	▪ Fault Detection ▪ Append Processor ▪ Real-time State Monitoring

DPTA is a method allocating and reallocating task according to the ratio of performance by considering RP performance. Performance index use CPU speed which

are the static resource factor of RP and job history. Assuming total job amount for the application is $TotJobSize$ and CPU speed value of RPs is $\{sRP_0, sRP_1, \cdots, sRP_{NumRP-1}\}$, the job size of random sRP_i is allocated as follows.

$$JobRP_i = \frac{sRP_i}{\sum_{k=0}^{NumRP-1} sRP_k} \times TotJobSize, \qquad i = 0 \sim NumRP - 1$$

The factor of job history is the factor measuring expectation of each RP performance and dynamically expects the job size per performance. The size of reallocated job for RP is as follows.

$$\mathrm{Re}\, AllocJobRP_i = pRateIncom\ RP_i \times (JobIncomRP_j - ComJobInco\ mRP_j)$$

$$pRateIncom\ RP_j = \frac{sComRP_i}{sComRP_i + sIncomRP_j \times \frac{ComJobRP_j}{sComRP_j}}$$

where, $sComRP_i$: Finished RP performance for the allocated job

$sIncomRP_j$: Unfinished RP performance for the allocated job

$JobsIncomR\ P_j$: Unfinished size of initially allocated RP job for allocated job

$ComJobsInc\ omRP_j$: Size of job performed by RP which did not finish the allocated job

$pRateIncom\ RP_j$: RP performance index occurring performance change

APTA is a method which monitors the static and dynamic performance of RP and allocates the job adaptively. This establishes a standard based on the performance executing real allocated task not a physical performance for the RP performance. RP performance value applies proportional factor and CPU speed. Proportional Factor(PF) is a standard value as CPU usage rate used by RP user or internal system differs in the degree of total execution time depending on the range. Table 4 is the proportional factor in different section.

Table 4. Proportional Factor in different section

Section	User CPU Usage	PF	Section	User CPU Usage	PF
A	0~10%	2.10	B	10~20%	1.91
C	20~30%	1.67	D	30~40%	1.40
E	40~60%	1.09	F	60% over	1.00

Assuming CPU speed value of RPs is $\{sRP_0, sRP_1, \cdots, sRP_{NumRP-1}\}$, the expectation of total CPU usage is $CPUUsageAv\,g$ at the time of reallocation and expectation of job processor usage rate is $JobUsageAvg$, the performance value of random RP_i is as follows.

$$RP_i Performance = (100 - (CPUUsageAvg - JobUsageAvg)) \times PF \times sRP_i \times 0.01$$

APTA uses RPPerformance value not the job history information and its reallocation is similar to DPTA. Reallocated job size of RP is as follows.

$$\mathrm{Re}\, AllocJobRP_i = pRateIncomRP_i \times (JobIncomRP_j - ComJobIncomRP_j)$$

$$pRateIncomRP_i = \frac{pComRP_i}{pComRP_i + pIncomRP_j \times \frac{ComJobIncomRP_j}{JobIncomRP_j}}$$

where, $pComRP_i$: value of RP which finished the allocated job

$pIncomRP_j$: value of RP which did not finish the allocated job

Fig. 4 is the comparison of task allocation method that user gives random CPU usage rate to 4 RPs with different specification. For the case if user CPU usage rate is below 10%, there is no difference in task allocation methods. This implies that performance changes almost did not in RPs and it did not influence on the task performance rate. On the contrary, the case of user CPU usage rate increasing to 20~30% and 30~40%, the total job performance time of APTA considering total CUP usage rate is low. That is, user CPU usage rate of RP influenced on job performance rate of total CPU usage rate.

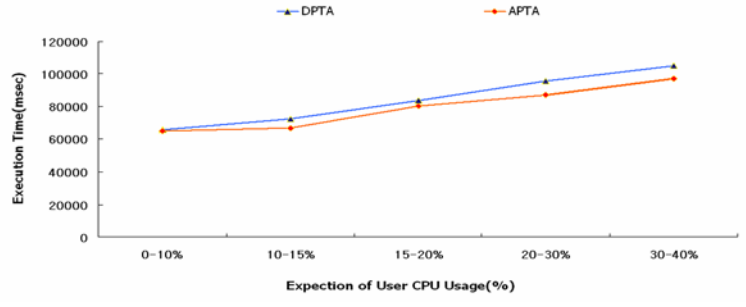


Fig. 4. Comparison of task allocation method according to the change of user CPU usage rate

5.2 GRMS Monitoring Information Visualization

The monitoring of RPs has effect on LRM scheduling and task allocation method which performs real operation. The state value of RP is measured and improved the operation usage and controlled the defect. After revising the metadata information, the state information is received through RP virtual observer and transferred to RP Entry and provided the user selection event with a view.

Fig. 5 provides the state information and analysis result of RP with one frame and five views. LRM provides task progress which is a performing task with progress bar component, and provides RP list which is connected to LRM with combo box. RP information view represents the selected metadata information of RP with progress bar and text component and applies to performance test of RP with analyzing sampled hardware information. RP Real-time Table View revised the RP information in a certain period and is represented in Table. RP state graph View represents the RP CPU usage and user CPU usage in the real-time graph form. Connection Graph View observes task control situation of RP which was connected to LRM. RP Network Information View represents the ratio between whole period of task control and network transmitting period of RP in the form of bar graph.

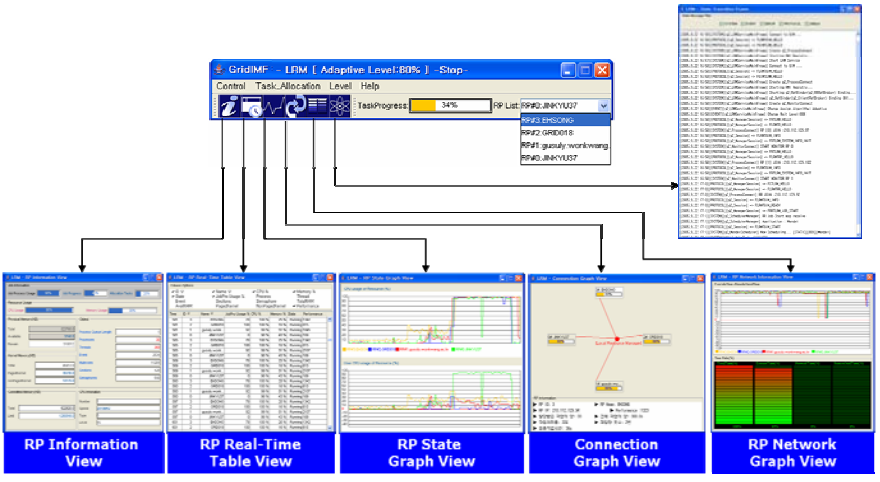


Fig. 5. Information Visualization View of GRMS

6 GridIMF Implementation and Performance Result

6.1 Application Architecture

Application is implemented with the verification of strategy suggested in GridIMF. Grid user has different request depending on application so that the application architecture is structurally divided only with minimum correlation.

For the mechanism minimizing interdependence between different application and GridIMF, Application Proxy is designed as shown in Fig. 6. Application Proxy is a standardized common API and includes core function for interacting between application and GridIMF. Application proxy is an Application Name of plug-in for accessing specific application and the object is generated dynamically by Application Templet.

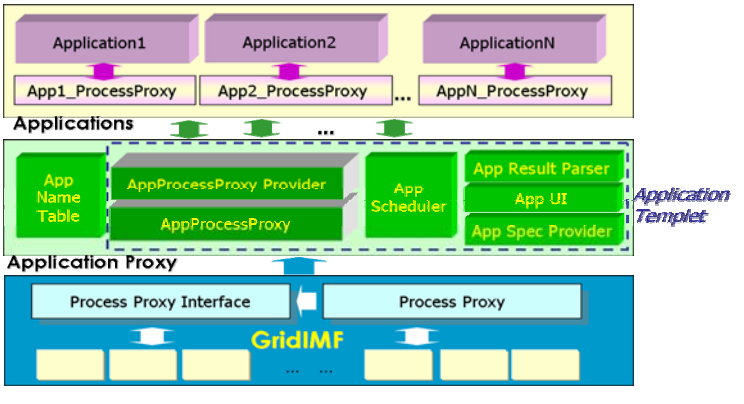


Fig. 6. Application Architecture of GridIMF

6.2 Implementation of Application and Result of Performance

The available application to GridIMF is independent to each task and it is supposed to require a lot of calculation compared to small amount of data. In this paper, the fractal image processing is shown as Fig 7. Application performs image generation by complicated calculation and performance process is certified visually. Fig. 3 (a) is the Mandelbrot fractal image processing of LSM coloring technique. Scheduling is APTA and two new RPs are joined in the calculation orderly in the middle of task performance by two RPs. Fig. 3 (c) is the result of the first calculation and there is a difference in coloring technique for re-assigning occurrence according to RP add and performance change. Resource usage and requirement change should be available before the end of GridIMF by grid user. So, Fig. 3 (c) can perform the re-requested job with changing of application list when the specific domain is established by drag. Fig. 3 (e) is Mandelbrot Fractal Image Processing which performs in dividing task according to the number of divergence. 300 times of divergency is supposed and the number of divergency is defined by task and it is distributed and performed to RP as performance ratio. Panel is divided into canvas which visualizes task management process and task result.

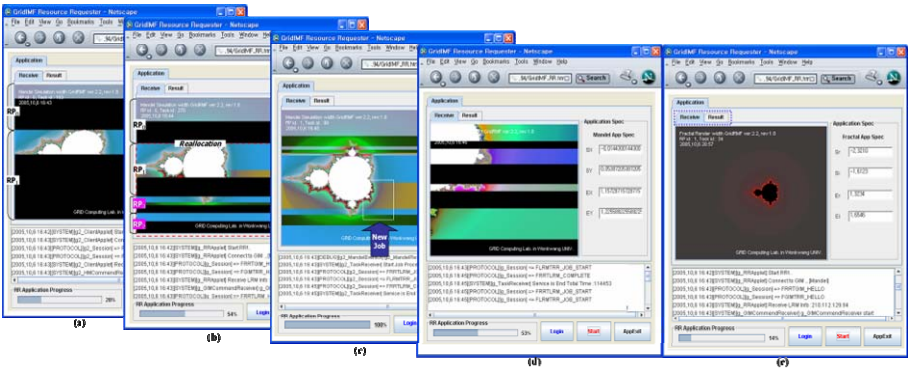


Fig. 7. Grid application adaptation of two fractal image processing with performance analyzing of GridIMF

7 Conclusions

In this paper, grid application was applied with constructing GridIMF to adapt grid Information and to provide the consistency to user. Grid information is a generic name of virtual organization, resource, and metadata. GridIMF is composed of 3-layer framework model such as infra layer for grid information, middleware layer for management, and application layer for application. Especially, middleware layer was divided into hierarchical 3-tier information management model according to participation intention, role, and management policy of grid information; and management domain was divided into GVMS and GRMS. GVMS gives authority to use remote resource through resource connection rule and task brokering with the optimal virtual organization selection mechanism by performance ratio. Even when the virtual

organization defect is occurred, LRM auto-recovery strategy is suggested to change the user requirement and to get rid of operation replication. GRMS supported the load balancing and fault tolerance with the adaptive performance-based task allocation which can control the grid information state change, participation, session, and defect. The information state monitoring and visualization of GridIMF supports the reliability of grid information collection. Application proxy was established for providing service which agreed to special grid application with the minimal code modification and the minimal relation with grid infrastructure. The service validity which was suggested in this paper was verified through grid application adaptation of two fractal image processing with performance analyzing of GridIMF and grid application. The main contributions of this paper are as followed; scalability (hierarchical 3-tier information management structure), adaptive (dynamic virtual organization, task allocation), availability of service, deletion of replication operation, information state monitoring/ visualization and independent of applications.

For further study, in the view of GridIMF approach which was suggested in this paper, skill add of user approach interface following detailed task process for user-centered task performance and dependability of detailed task and adaptation of specific application are needed.

Acknowledgement. This paper was supported by Wonkwang University in 2006.

References

1. I. Foster, C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmannn, 1999.
2. I. Foster, C. Kesselman, S. Tueche, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International Journal Supercomputer Applications, Vol. 15, No. 3, 2001.
3. I. Foster, "The Grid: A New Infrastructure for 21st Century Science," Physics Today.org, 2002.
4. A. Takefusa, H. Casanova, and S. Matsuoka, F. Berman, "A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid," High Performance Distributed Computing, 2001. Proceedings 10th IEEE International Symposium, Aug. 2001
5. Dr. Bernhard R. Katzy, "Design and Implementation of Virtual Organizations," University St. Gallen and Erasmus University Rotterdam.
6. Rashmi Bajaj and Dharma P. Agrawal, Fellow, "Improving Scheduling of Tasks in a Heterogeneous Environment," IEEE Trans. Parallel and Distributed Systems, Vol. 15, No. 2, pp. 107-118, 2004.
7. Rajkumar Buyya, Steve Chapin, and David DiNucci, "Architectural Models for Resource Management in the Grid," Grid Computing GIRD 2000. First IEEE/ACM International Workshop Bangalore, India, pp. 20-33, 2000.
8. I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit, " International Journal supercomputer Applications, Vol. 11, No. 2, pp. 115-128, 1997.
9. D. Angulo, I. Foster, C. Liu, and L. Yang., "Design and Evaluation of a Resource Selection Framework for Grid Applications," Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 2002.

Framework for Enabling Highly Available Distributed Applications for Utility Computing

J. Lakshmi[1], S.K. Nandy[1], Ranjani Narayan[2,⋆], and Keshavan Varadarajan[1]

[1] Indian Institute of Science Bangalore, India
{jlakshmi@serc, nandy@serc, keshavan@cadl}.iisc.ernet.in
[2] Morphing Machines, Bangalore, India
ranjani.narayan@morphingmachines.com

Abstract. The move towards IT outsourcing is the first step towards an environment where compute infrastructure is treated as a service. In utility computing this IT service has to honor Service Level Agreements (SLA) in order to meet the desired Quality of Service (QoS) guarantees. Such an environment requires reliable services in order to maximize the utilization of the resources and to decrease the Total Cost of Ownership (TCO). Such reliability cannot come at the cost of resource duplication, since it increases the TCO of the data center and hence the cost per compute unit. We, in this paper, look into aspects of projecting impact of hardware failures on the SLAs and techniques required to take proactive recovery steps in case of a predicted failure. By maintaining health vectors of all hardware and system resources, we predict the failure probability of resources based on observed hardware errors/failure events, at runtime. This inturn influences an availability aware middleware to take proactive action (even before the application is affected in case the system and the application have low recoverability).

The proposed framework has been prototyped on a system running HP-UX. Our offline analysis of the prediction system on hardware error logs indicate no more than 10% false positives. This work to the best of our knowledge is the first of its kind to perform an end-to-end analysis of the impact of a hardware fault on application SLAs, in a live system.

1 Introduction

The move towards IT outsourcing is the first step towards realization of Utility Computing[1] [2], where compute infrastructure is treated as a service. The IT service in itself comprises the hardware (servers, storage, etc.), the system software (Operating system enabling the use of hardware) and distributed software (Middleware enabling the use of distributed systems). Dependability of the IT service is its capability to deliver service that can be trusted irrespective of failures either due to faults, malfunction or security breaches. Availability, reliability, safety, confidentiality, integrity and maintainability are the attributes

⋆ At the time of submission the author was an employee of Hewlett-Packard ISO. The author's affiliation has changed since Aug 1, 2006.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 549–560, 2006.
© Springer-Verlag Berlin Heidelberg 2006

of dependability [3]. Depending on the application using the system, some or all of these attributes play a significant role. Traditionally, elaborate proprietary solutions, using redundancy options with resource and service replication, were adopted to ensure reliability. In such solutions cost itself is a limiting factor. With increasing globalization, such reliability cannot come at the cost of resource duplication, since it increases the TCO of the compute infrastructure and hence the cost per compute unit. Today, with advances and standardization efforts in Internet technologies many distributed collaborative applications are being seriously explored.

Distributed systems have inherent resource replication due to several under utilized and available systems. Business houses could use these resources as redundant servers effectively, by judiciously deploying business applications. The problem appears when a system or a service fails in such distributed environments. At the site of failure the faults maybe visible, but are not appropriately propagated to the handling application because of which the error-handling strategies may not be effective. A simple example to illustrate this point is that of the common I/O errors encountered while writing to or reading from a file due to disk media errors. The local machine's Operating System (OS) has knowledge of the media errors and if it is smart would issue block relocation transparently without the application even knowing of the error. If the block relocation at the disk level is not successful, the OS could return equivalent error message that could be trapped by the middleware, which in-turn could relocate the file to a different file system. In both cases the application may experience latency but not errors. Such proactive recovery strategies play a useful role, particularly in distributed environments spreading across multiple administrative domains where most faults leave signatures on their local hosts, which are not visible or accessible to remote applications.

In this paper, we explore such scenarios with specific focus into the aspect of hardware failures effect on highly available distributed applications and the techniques required to take proactive recovery steps. We propose a method for projecting hardware failures, occurring or predicted to occur, in terms of the resources being used by an application process, at runtime, and use the projections to take proactive recovery steps with a view towards maintaining QoS guarantees like availability. We describe an end-to-end distributed framework to support proactive recovery and detail the functionality of each of the framework's components.

2 Review of Related Work

Current state of fault management for distributed systems is addressed disparately at three different layers, namely hardware (ECC and CRC like checks, RAID, etc. associated with hardware redundancy), system software (OS level resource duplication, reallocation strategies) and application (checkpoint and rollback recovery, replication, backup strategies).

Hardware Fault Tolerant techniques. The existing literature on fault tolerance has many reported techniques to mask hardware failures. The basic strategy used is to contain errors and this is usually achieved by establishing ECC checks. Once an error is identified and cannot be rectified some form of redundancy is used to mask the failure. Many hardware redundancy techniques, ranging from component duplication, circuit-level duplication to board level duplication, have been adopted [4]. All these methods are capable of identifying problems at the system hardware and effectively mask hardware failures. The OS may or may not be designed to exploit these features. Also, provisioning for hardware level fault tolerance usually tends to be a costly affair.

Operating System Fault Tolerant Techniques. The OS abstracts the hardware and provides a usable interface to the hardware devices. As a consequence, what the application process uses as a resource is quite different from what is available as a physical device. Most error monitoring mechanisms at the hardware level, work at the physical device level. Although OS is normally aware of the errors occurring at the hardware level, it still needs to map the physical devices to the abstracted resources to really diagnose what resource of the currently active process(es) is affected. As of now no OS does this diagnosis. Most OSs just report the hardware error with the associated physical device, and in some cases the affected OS abstracted resource, and leave it to the diagnostics personnel to analyze and solve the problem. Usually the running process affected due to an error is aborted. Recently, IRON File systems [5] work cites this deficiency and suggests some form of recovery for disk-based errors on the file resources a process uses. We also believe that similar recovery strategies can be employed for masking errors of other hardware devices like memory, network and I/O interfaces.

Application Level Fault Tolerant techniques. Application level fault tolerance is the most commonly adapted technique since it is easier for the developer to implement. Most strategies fall into the category of check-pointing with roll-forward or roll-backward (restart strategy), n-version programming (replication strategy) or self-checking software [6] [7] [8]. Most of these methods are quite expensive and heavyweight particularly when applied to distributed applications. Often, most of the above-mentioned techniques are associated with highly available enterprise class systems. These systems have fairly excessive monitoring hardware and software, but the recovery operations are restricted to application level. Many a times, particularly with respect to hardware faults, application recovery is coarse grained. For example, in distributed environments like the Computational Grids, an application is set for execution after the required input and execution files are staged-in to the chosen execution host. If the file-staging area is hosted on a disk, which is experiencing media errors, the application execution may fail. A typical recovery option applied when the application fails is either restart or migrate to another host. This migration tends to be costly if it is initiated after the execution has progressed to some extent. If there is a mechanism that can detect why the execution failed, like in this case due to disk media errors, the required failed file could be staged-in, on the fly, to a

stable storage area, free of errors. The application file I/O request could be re-initiated transparently. This needs error monitoring and recovery capability at all stages, from the hardware, to the OS to the application, as one coordinated effort. In order to enable such fine-grained recovery one needs to 1. Identify errors occurring in the system at runtime 2. Characterize the failure effect on the active processes currently running in the system 3. Devise and apply multi-level recovery strategies to mask failures. In this study, we have tried addressing the effect of hardware failures on the running processes in a system and propose a framework that monitors and allows for multi-level recovery.

3 End-to-End Architecture for Fault Tolerance in Distributed Environments

In distributed setups, an application can potentially get realized over physically different machines, each possibly contributing in a different way. In its simplest form the application would have three parts, namely, the client part from where the user launches his request, the middleware part that understands what services are required to satisfy the user request and the resource part that actually executes the services. Typically, such applications have multi-tier architecture as shown in Figure 1.

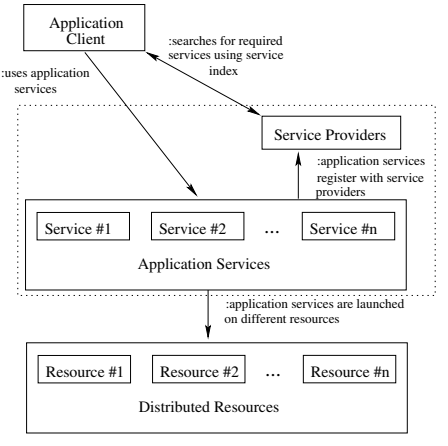


Fig. 1. Multi-tier architecture of a distributed application

The Application Client is the client part, the Service Provider along-with Application Services description and access information (dotted box in Figure 1) form the middleware part and the Distributed Resources form the server part. A distributed application has client and server components. The client hosts the application request launcher and the server hosts the services that the application uses to satisfy user requests. In collaborative service environments,

the service composition for a specific user request could happen dynamically at runtime. In order to locate the currently available services, application clients communicate with known service providers. The service providers play important role in locating services that satisfy specific QoS requirements, as stated by the client request. Within distributed environments, this means that every component participating in satisfying the request contributes to the QoS. In order to assess satisfaction of QoS guarantees, appropriate monitors are established, usually at each layer of the multi-tier. What QoS requirements are specified along-with the request depends on the application and the specific request and is monitored by the service provider through a Service Level Agreement (SLA). This SLA is composed of QoS specific metrics that are monitored. These metrics are either associated with service, resource both. In order to ensure QoS guarantee satisfaction, one can use resource reservation followed by allocation schemes and then restrict system load to prevent violations. Adaptive schemes could also be used which could allocate more than requested resource quantity, like communication bandwidth, when there is lesser load [9]. All schemes concentrate on handling QoS guarantees when multiple applications are contending for resources, but do not address the issue of non-satisfaction of QoS guarantees in face of resource failures. In order to do so, like any other QoS metric, one should be able to measure as well as monitor resource failures, their effect leading to service failures and thereby resulting in application failure. Here we describe one such framework for highly available applications, that is designed to monitor resource failures and their effects on services and the application, with an aim to enable pro-active recovery. The basic input for the monitoring components in this framework is availability. For the purpose of discussion in this paper we define availability as the probability that the required resource or service is available at a given point in time to render the requested service. As of this study we have restricted to measuring availability with respect to hardware component failures in a system. Also, we define, two health vectors in order to monitor availability namely hardware health vector and process health vector.

We define the hardware health vector as the list of availability probability values of all the hardware components of a system. Typically, this would contain the availability probability values for processor, memory, disks, network cards, I/O cards, etc. In a Unix based machine these are usually the physical devices of the system and are identified by a unique device path name, like the device hardware path in HP-UX (HP's implementation of Unix).

Any process executing on a system, uses the system's software services and the hardware devices to carry out its function. Assuming that the software part of the executing process and the system are fault-free, we map the hardware failures to the using process based on its current resource usage. We define this mapping as the process health vector. The process health vector contains a list of the availability probability values for the resources it is using or expected to use. Typically this would contain the values for the files and sockets the process needs, the virtual memory pages and swap area blocks it is currently using,

multiple threads it has spawned, etc. We derive the process health vector from the hardware health vector as described in the later section.

Having defined the hardware and process health vectors, we now describe the framework showing how these vectors can be used to detect failures and activate recovery. It is useful to note here that any pro-active mechanism would benefit if a prediction of the failure is made prior to the actual failure. The prediction gives the proactive mechanism time to react. Of course, none of this is effective in case of crash failures wherein reactive measures need to be applied.

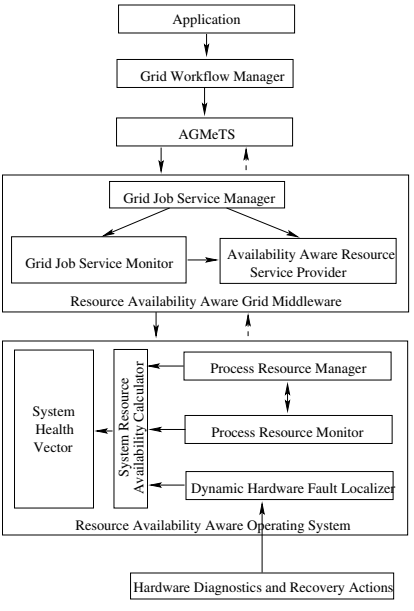


Fig. 2. Pro-Active Fault Management Framework for Distributed Environments

Figure 2 shows a schematic diagram of the framework. The diagram depicts a typical multi-tier architecture of distributed environments like the Computational Grid [10]. We use the Computational Grid as a representative example for describing the framework. However, the framework is not restricted to Grids and is generic enough to be applicable to any distributed environments. In this figure there are four-layers, namely, Grid user integrated development environment (Grid IDE) [10], Adaptive Grid Meta-Scheduler (AGMeTS) [11], Grid Middleware [10] and the Grid Fabric [10], in order of flow of execution control. A Grid user composes his application and states it as a workflow composed of job steps using the Grid IDE. Each job step could be an independent job composed of collaborating services. A job step is specified with its resource and QoS requirements. Resource availability is specified by the job step as a QoS requirement. A highly available application would specify its availability requirement as 100% resource availability. For example, the job step specifies its resources requirements as:

```
<ncpus=1, memory=1Gb, input_file="/home/myinput",
 output_file="/home/myouput", scratch_dir="/home/scratch">
```

and required resource availability as:

```
<processor=1.0, memory=1.0, input_file=1.0,
 output_file=1.0, scratch_dir=1.0>
```

The value of 1.0 as availability number against a resource means that the application expects the resource to be 100% available, at all times, during its execution run. An application that wants a best of effort run need not specify availability as a QoS requirement. AGMeTS, Grid Middleware and the Grid Fabric use the availability information specified by the job step as an input for making scheduling, monitoring and recovery decisions. AGMeTS [11] combines resource brokering with fault tolerance as it's goals and is successful in masking resource failures autonomously in distributed environments. AGMetS is an intelligent distributed job scheduler that takes scheduling decisions based on the knowledge of the system availability. We expand the system availability to include the process health vector and AGMeTS to react to changes in this vector.

After the user completes the composition of his application, the Workflow manager calculates each job step's dependencies, composes an execution plan and submits each job step according to this plan to AGMeTS. AGMeTS parses the resource and availability requirements of the application and selects a suitable compute node for dispatching the job. AGMeTS uses the job submission service of the Grid Middleware for this. Solid arrows depict this flow of execution control in Figure 2. Once the job step is in execution, AGMeTS starts monitoring the host's resource availability against the QoS requirements specified for the job step. AGMeTS gets this information from the Grid Middleware's resource service provider. Dashed arrows in Figure 2 depict this flow of information.

The job submission service of the Grid middleware sets up the Grid Job Service Monitor and Manager services after the job has been successfully submitted to the compute node. If the current host's resource availability drops below acceptable limit, AGMeTS triggers job-step recovery action like issuing an application checkpoint and job step migration to the next suitable node. The Grid Middleware publishes the host's resource availability information specific to the job step, the process health vector. The Job service monitor uses this information to watch current availability of resources for the job step. In case a specific resource's availability drops below acceptable limits, the monitor triggers resource specific recovery action. For example, the compute node detects file read error on the job step's *input file*, after the time it has started reading, due to media errors on the file staging disk. At this point in time the job-step would fail with an I/O error on *input_file*.

Since now the compute node is publishing loss of availability on the *input_file*, the Grid Job Service Manager can take an autonomous decision on relocating the file by issuing GridFTP afresh. The process resource manager running on compute node would request the OS of the compute node to relocate the file from the recent transfer to an area on disk that has so far not experienced any media errors or locate it to an alternate disk.

In this framework, we use the concept of recovery oriented computing [12] to incorporate autonomous behavior, in face of failures, to the application. The necessary information to take decisions on when to start recovery is provided by the health vectors. These health vectors are interdependent and composed at runtime. The interdependence is bottom-up, i.e., once we know the compute node specific hardware resources (like disk, processor, etc.) availability, we can map the process' resources (like thread, file, stack, etc.) health to that of the hosting hardware resources and from these compute the host's health on which the job-step is executing.

4 Generating the Hardware Health Vector

The hardware health vector of a system can be generated using any online error monitoring and analysis method. For our purpose we use an analysis method built using Bayesian Belief Network (BBN) [13]. Based on the current observed hardware error events, the BBN outputs the failure probabilities of the effected components from which the hardware health vector is easily computed. We also use a standard exponential decay function for each hardware component to make a prediction on its future health knowing the current health. This method was tested on HP-UX based machines. Our offline analysis of the prediction system on hardware error logs indicate no more than 10% false positives. What is important here is the quantification of the availability values, which helps in rendering the property of autonomous behaviour. Quantification enables agents or monitors to check the values periodically or based on thresholds and take appropriate decisions, like pro-active recovery. For one of the system used for job submission, a failing disk was found. The hardware health vector computed for the node was as follows:

```
<node_name: oldmonk>
<current>
<processor:32:1.0 0.0> <memory:49:1.0 0.0> <scsi_adapter:10/0/15/0:1.0 0.0>
<disk_controller:10/0/15/0.6:1.0 0.0>
<disk:10/0/15/0.6.0:0.013 0.987:block_address:>
<predicted:02.00.00>
<processor:32:1.0 0.0> <memory:49:1.0 0.0> <scsi_adapter:10/0/15/0:1.0 0.0>
<disk_controller:10/0/15/0.6:1.0 0.0> <disk:10/0/15/0.6.0:0.013 0.987>
```

The node named "oldmonk" is a single Intel processor based machine with 256MB memory with a single hard disk of 80Gb capacity connected to a SCSI adapter. In the vector above, followed by the node name is the current health of the component list constituting the component generic name, hardware path (unique device name in the system) along-with it's working and failure probability values. For devices like disk, additional information like block address is given, in case where media errors are detected. The block address of the failing block is useful to locate the actual affected resource, in this case a file, while computing the process health vector. Following the current health vector, the predicted component health values after a period of 2 hours is listed.

5 Generating the Process Health Vector

The hardware health vector is generated based on observed hardware error events in the system. Observing these events one discovers that these events relate to the hardware component's specified functioning. At a process level, the OS abstracts the hardware devices into appropriate resources like virtual memory pages, input-output files, etc., that are usable by the process. While computing the health vector at the process level one needs to get the correct associations of the involved hardware devices to the abstracted OS resources. For the example cited earlier for the disk failure, we use the disk block address to verify if the concerned files, currently allocated to the process, contain the block address. If so, then the loss of availability due to the error on the disk block is factored in while calculating the availability of the file. Before actually projecting the loss of availability for the file, we envisage to instrument appropriate mechanisms in the file system manager of the OS to dynamically relocate the file block to another disk block. If this operation is successful, there is no loss of availability projected for the file. In case this is not successful, the local OS could suspend the process, and if the application allows for change in the file pathname, relocate file to a different file system and restart the process. In this case also, there is no loss of availability projected for the file. The application continues to execute after experiencing latencies rather than terminating. In case, the file pathname cannot change, the loss in file availability is projected in the process health vector. This process health vector is monitored by the AGMeTS which would take application level recovery action when the process resource availability levels drop to unacceptable values.

Since this method was prototyped on HP-UX OS, the following two figures, Figure 3 and Figure 4 give details of generating file resource availability in the process health vector from the disk subsystem hardware health vector. The diagram contains HP-UX related OS internal data structures taken from [14].

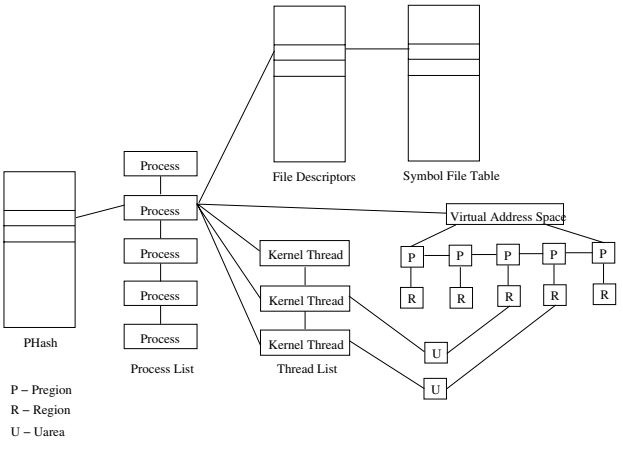


Fig. 3. Kernel Process Tables in HP-UX (version 11i)

Figure 3 above gives the details of data contained in the HP-UX kernel's process tables. The *phash* function in the above diagram is a hash table on the process id to the *proc-list* structure of HPUX. The *proc-list* contains details of all the current active processes in the system. These details include the number of threads spawned by the process, its virtual memory extents, file descriptor entries, etc. For every process of interest, we find out the resources it is allocated by navigating through these tables and form the process health vector elements. An example process health vector, for the job requirements specified in section 3 is as follows:

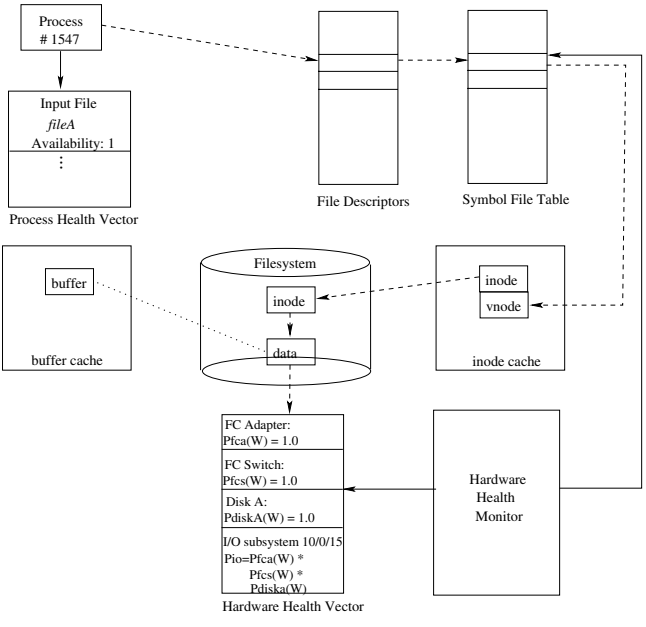


Fig. 4. HP-UX view of a filesystem tables along-with the mapping of the hardware health vector to the corresponding resource element of the process health vector. P(W) for a hardware component depicts the probability of the component working at a given point in time.

```
<job_id:1547>
<processor=1.0, memory=1.0, input_file:fileA=1.0,
 output_file:fileB=1.0, scratch_file:fileC=1.0>
```

For each of the vector elements' we then navigate the appropriate kernel resource table for a mapping between the hardware health vector to the specific vector element of the process health vector. Figure 4 depicts the mapping between the input file *fileA* for the process *1547* and the underlying hardware this resource is realized on. In this specific case, the file *fileA* was physically located on a disk *DiskA*. Incidentally, in this case the filesystem on which this file resides, was

located only on this one disk. In order to consider the availability of the *fileA* we need to consider it's dependency and also the availability of the hardware of the I/O subsystem which the device driver interacts for any I/O operation on this file, the memory regions occupied by the inode and file system buffer caches and the mount points. As of this study, mapping between an I/O subsystem to the file is complete. We need to extend the mapping to include the memory and mount point dependencies.

The *proc_list* for the process *1547* gives the file descriptor for *fileA*, which is added as an element to the process health vector for this process. From the system file table we can trace through the inode cache to the specific inode and the file data block. From the inode and the data blocks for the file we get to know the disk block extents occupied by the *fileA*. The hardware health vector generated for the I/O subsystem containing this *DiskA* contains the current hardware health status of the disk as well as the complete subsystem. While calculating the availability of the *fileA* we use the subsystem availability value since any hardware element within the subsystem, including the disk, would contribute to the file's failure. This is depicted in Figure 4.

6 Conclusion

The framework described in this paper facilitates adaptive recovery, based on observed resource failures. The framework is end-to-end and builds on multi-level recovery approaches to mask failures. It benefits from this approach since at the lowest level, i.e. hardware level, the recovery is lightweight. Increasing complexity of recovery is attempted only when the lower level techniques fail. This is against the most common approaches adopted today that tend to either restart with checkpoints or replicate the application. This framework uses such methods only when other recovery options have failed. This work to the best of our knowledge is the first of its kind to perform an end-to-end analysis of the impact of a hardware fault on application SLAs, in a live system. The use of quantified current and predicted health aids in making appropriate decisions on the type of recovery performed through the use of policy engines.

We are in the process of evaluating the performance overheads on the application. Since we are looking at application failure scenarios, and the strategy is to provide for fault masking at various layers of the framework, we believe that the application does not really have any significant performance loss. The benefit cannot be worse than the case where manual intervention is the only option in Grid like distributed environments. We are also exploring the use of virtual machines, like Xen and Vmware, with features like live application migration and relocation, wherein such pro-active methods are meaningful and can help the middleware to decide when to migrate the application alongwith it's state.

Acknowledgments. The authors thank Harshit Singh, Tushar Agrawal and Vagdevi Kudlur for prototype implementation and also Seenivasagan Manavalam and his team of Hewlett-Packard India Pvt. Ltd. for providing us with

representative hardware error logs. This work was supported in part by Hewlett-Packard India Software Operations.

References

1. Ross, J.W., Westerman, G.: Preparing for utility computing: The role of it architecture and relationship management. IBM Syst. J. **43** (2004) 5–19
2. Sahai, A., Singhal, S., Joshi, R., Machiraju, V.: Automated policy-based resource construction in utility computing environments. In: IEEE/IFIP Network Operations and Management Symposium. (2004) 381–393
3. Lussier, B., Chatila, R., Ingrand, F., Killijian, M., Powell, D.: On fault tolerance and robustness in autonomous systems. in proceedings of the 3rd IARP-IEEE/RAS-EURON joint workshop on technical challenges for dependable robots in human environments, manchester (gb), 7-9 september 2004 (2004)
4. Erez, M., Jayasena, N., Knight, T.J., Dally, W.J.: Fault tolerance techniques for the merrimac streaming supercomputer. In: SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing, Washington, DC, USA, IEEE Computer Society (2005) 29
5. Prabhakaran, V., Bairavasundaram, L.N., Agrawal, N., Gunawi, H.S., Arpaci-Dusseau, A.C., Arpaci-Dusseau, R.H.: Iron file systems. In: SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles, New York, NY, USA, ACM Press (2005) 206–220
6. Hwang, S., Kesselman, C.: Gridworkflow: A flexible failure handling framework for the grid. In: HPDC '03: Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing (HPDC'03), Washington, DC, USA, IEEE Computer Society (2003) 126
7. Zhang, X., Zagorodnov, D., Hiltunen, M., Marzullo, K., Schlichting, R.: Fault-tolerant grid services using primarybackup: Feasibility and performance (2004)
8. Xie, Z., Sun, H., Saluja, K.: Survey of fault-tolerant techniques in modern microprocessors. Technical report, (Department of Electrical and Computer Engineering, University of Wisconsin-Madison)
9. Foster, I., Kesselman, C., Lee, C., Lindell, R., Nahrstedt, K., Roy, A.: A distributed resource management architecture that supports advance reservations and co-allocation. In: Proceedings of the International Workshop on Quality of Service. (1999)
10. Foster, I., Kesselman, C.: The Grid - Blueprint for a New Computing Infrastructure, second edition. Elsevier Publication (2004)
11. Nainwal, K.C., Lakshmi, J., Nandy, S.K., Narayan, R., Varadarajan, K.: A framework for QoS adaptive grid meta scheduling. In: DEXA Workshops. (2005) 292–296
12. Patterson, D., Brown, A., Broadwell, P., Candea, G., Chen, M., Cutler, J., Enriquez, P., A. Fox, E.K., Merzbacher, M., Oppenheimer, D., Sastry, N., Tetzlaff, W., Traupman, J., Treuhaft, N.: Recovery oriented computing (roc): Motivation, definition, techniques, and case studies. Technical Report UCB//CSD-021175, UC Berkeley Computer Science (2002)
13. Narayan, R., Varadarajan, K., Lakshmi, J., S.K.Nandy, Agrawal, T., Singh, H.: Fault localization and recovery engine: A learning system. In: Hewlett-Packard Technical Conference. (2005)
14. Cooper, C., Moore, C.: HP-UX 11i internals, first edition. Prentice Hall Publication (2004)

A Bluetooth MPI Framework for Collaborative Computer Graphics

Daniel C. Doolan[1], Sabin Tabirca[1], and Laurence T. Yang[2]

[1] University College Cork, Ireland
d.doolan@cs.ucc.ie, tabirca@cs.ucc.ie
[2] St. Francis Xavier University Antigonish, B2G 2W5, Canada
lyang@stfx.ca

Abstract. This paper introduces a collaborative framework that resides on top of the Bluetooth API. It is designed for Bluetooth enabled mobile devices to allow for the collaborative generation of computer graphics and message passing. This new library is called the Mobile Message Passing Interface (MMPI). Mobile devices generally have limited processing abilities. By combining the processing power of several devices, complex computer graphics can be rendered in a fraction of the time a single device would take.

1 Introduction

The uptake of Mobile phones around the world is staggering with forecasts of 810 million units shipped for 2005 [13]. This is close to one sixth of the worlds population in just a single year. Compare this with the expected sales of PC's at 200 million units for 2005. "What would you call a device that has a screen, a keyboard, storage, email, documents, the ability to play audio and video, games spreadsheets, and communications ability? A personal computer? How about a 'mobile phone'?" [15]. One can clearly say from the previous statement that mobile phones have most of the computing abilities required by the public.

Current top of the range phones are running at about 100 - 200Mhz, the phones within the next few years will be as powerful as many personal computers that are in use today. On the 4^{th} October 2005 the Cortex-A8 processor was unveiled, yielding processing power of 1Ghz [24] [1] [2]. This is the type of processing power that many devices such as phones, PDA's, digital cameras and TV's will have in the not too distant future.

It is clear that many mobile devices (phones, PDA's) have potential for computational tasks. As with any large task, adding more processing units to the problem generally helps to reduce to overall computation time. This is where Bluetooth technology can be of use. So what is Bluetooth? In short it is a low power short range communication system, named for Harald Blåtand "Bluetooth" II, King of Denmark from 940 to 981. He was renowned for getting people to talk to each other. Hence the modern usage of the term "Bluetooth" for the area of short-range communication.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 561–572, 2006.
© Springer-Verlag Berlin Heidelberg 2006

1.1 Article Overview

Several distinct areas are discussed within this paper as summarised below.

- Section 2. Discusses Message Passing and Bluetooth networking.
- Section 3. Details the structure of the MMPI library and gives some examples of the communication methods in use.
- Section 4. Shows how the library can be applied for the creation of Fractal Image in parallel and examines how various load balancing schemes affects the overall processing time.
- Section 5. Focuses on the use of the library for collaborative gaming.

1.2 Importance of Bluetooth and Mobile Devices

Bluetooth allows devices to communicate within a short range. For devices such as phones Bluetooth is a very useful means of communication as it has a very low power consumption. It is generally of use where a small amount of data needs to be transmitted. The Bluetooth 1.2 specification defined the maximum data rate to be 1 Mbits/s (723Kbit/s real throughput). The Enhanced Data Rate (EDR) version of Bluetooth was ratified in November of 2004. Under the Bluetooth 2.0 specification EDR operation will deliver a maximum of 3 Mbits/s (2.1 Mbits/s real throughput) [16].

Bluetooth has many uses: sending data over a modem, sending voice and GPS data are typical uses. Any Bluetooth enabled devices may be connected together. J2ME allows developers to program for Bluetooth via the JSR-82 package. Client / Server systems can be developed to carry out what ever operation the developer requires, from multiplayer games to connecting to database Servers.

1.3 The Need for Collaborative Computation

There are many problems that require extremely powerful machines to compute the answer in a reasonable time for example weather forecasting. Such problems cannot be tackled by a single system as it would take years of computing. The solution to this is to use many processors. Currently the most powerful system is IBM's BlueGene Project which consists of 131,072 processors (top500.org).

The largest distributed systems in the world use this paradigm. The Berkeley Open Infrastructure for Network Computing (BOINC) [4] has many research projects to which interested parties can contribute their systems computing power. These projects include: the study of climate change and the investigation of protein related diseases. The most famous and largest with well over 5 million participants is the Search for Extraterrestrial Intelligence (SETI) [5].

Mobile devices have far less processing power than that of desktop machines. Hence if a task requires a significant amount of time to compute on a single device then the answer is to distribute to work to multiple devices. Bluetooth provides a useful means of connecting several mobile devices together, forming the underlying infrastructure for collaborative computation with a set of mobile devices.

1.4 Motivation

Many cluster systems use the Message Passing System to communicate between nodes. It allows the user to write a single program that can run on multiple processors without the need of the developer to worry about the underlying communication mechanism.

The development any Bluetooth system inherently requires the programmer to write significant sections of Bluetooth specific code. The primary reason for developing the MMPI system is to abstract from the underlying Bluetooth communication system. This higher level of abstraction allows for shorter development cycles. The developer can focus on the application task at hand and not have to deal with the intricacies of inter device communication.

The MMPI system can be used within many application domains and is not just limited to high end parallel computation. It can also be used for multiplayer gaming and simplifies any application that requires Bluetooth communication.

1.5 Library Usage Intentions

One can say that Bluetooth communications is inherently a nomadic communication form where devices come and go from a network at random. The principle behind this library is that it is used in an environment where this nomadic behavior does not occur. The use of the library for gaming is an example were several individuals would setup a game and partake in it for a prolonged period of time. It may also be of use within the education environment to communicate questions and answers, even small exams within the class room. Many people no longer are the owners of just one phone, hence the library can prove to be a useful tool for synchronising data between several devices, be it phone, PDA or laptop.

2 Operating Environment / Background

2.1 Message Passing

Since it's introduction in 1992 parallel programming has long benefitted from the Message Passing Interface (MPI). What is MPI? "MPI is a library specification for message-passing, proposed as a standard by a broadly based committee of vendors, implementers and users" [21].

MPICH is one of the most well known freely available MPI systems currently available [22]. MPI systems are usually based on the C or Fortran programming languages. There are also Java based implementations available (mpiJava) [18], this is an object-orientated Java interface to the standard message passing interface. It is implemented as a set of Java Native Interface wrappers to native MPI packages. Another all Java example is Message Passing in Java (MPJ) [9] [3].

2.2 Bluetooth

Bluetooth [6] [7] is a wireless (radio) standard designed for short range communication and low power consumption. Bluetooth products are divided into one

of three classes based on communications power, allowing for omni-directional communication between devices ten to 100 meters apart. Communication is carried out in the Industrial-Scientific-Medical (ISM) band at 2.4Ghz. The Bluetooth operating band is divided into 79 channels each 1 Mhz apart (2.402 to 2.480Ghz). If also uses frequency hopping, changing channels 1,600 times per second.

2.3 Bluetooth Network Topologies

Bluetooth Networks can be established in three differing forms: Point to Point, Piconet and Scatternet. The simplest form (Point to Point) connects two Bluetooth enabled devices together. Typical uses of this form are: phone / PDA connections to a Keyboard or phone to Bluetooth Headset to name just two possibilities.

The Piconet is a network that may have up to eight devices connected together at any one time. In other words a device has a limit of seven connections that it can make to other devices.

The Scatternet is composed of a collection of Piconets. To connect the various Piconets together some of the devices must act as both Client and Server to establish the interconnect between separate Piconets.

3 Mobile Message Passing Interface (MMPI)

The primary purpose of MMPI is to produce an MPI like system for mobile devices. This will abstract the programmer from the Bluetooth communications libraries and allows for more productive development of the task at hand.

3.1 System Structure

The overall structure consists of three classes. Firstly the main MMPI class which carries out the primary message passing functions. The two remaining classes BTClient and BTServer are required for creating the underlying Bluetooth connections. The MMPI class will instantiate only one of the Bluetooth Classes depending on the parameter value that is sent to the constructor. With the channels of communication established the MMPI class is capable of sending or receiving messages simply by accessing an element in an array of either DataInputStreams or DataOutputStreams.

With the MMPI object instantiated the size and rank can be easily established by calling the getSize() and getRank() methods of the MMPI object. By knowing these values an application using the MMPI system can divide up the task appropriately in preparation for the communication of data to all nodes in the system.

To write any MMPI application a few simple lines of code are required. Between the creation of the MMPI object the the closing down of the interconnect (with the finalize() method) parallel computation may be carried out. In many programs one of the first methods to be called are getSize() and getRank() so the node has an idea about it's environment. This is followed by the parallel code the user wishes to execute.

3.2 Initialising the World

The startup of the inter-connect between all the nodes begins as a typical Bluetooth Client / Server system. Firstly several Client (slave) nodes are started up. Each of these open up a Server connection and then waits (blocks) until a Client connection is established. This is achieved via the openAndAccept() method. To create our world with five nodes requires the initialisation of four Client applications, each of which establishes a Server connection to await Client connections. Next the root node (Our Server) establishes connections to each of the Client nodes by carrying out both device discovery and service discovery. At the end of this process for five nodes we have an interconnect similar to that of Figure 1.

The main node has an array of DataOutputStreams and DataInputStreams to each of the connected Clients. With this one to many connection the main node can communicate with all of the connected Clients. However a system is required whereby any node can communicate with any other node in the system.

This procedure requires the setting up of several more Server and Client connections to complete the interconnect. The establishment of these connections must be synchronized so that the appropriate Server connection is connected with the correct Client connection. This synchronization process is carried out through the main node of the system. Firstly all of the connected Clients need the addresses of all the other nodes in the system. Currently the only address that a Client knows of is that of the main node (root node). So once the root node has established connections with all the Clients it transmits an array of strings representing the address of all the Clients in the system. With these addresses it is now possible to interconnect all Clients to each other.

Every device in the system requires the establishment of two arrays one for input the other for output. The index into this array should give a connection to the same device no matter what node is currently being examined. This means that we can establish communication in a similar fashion to MPI whereby the source and destination addresses are referenced by index numbers (the root node being 0). It is also necessary to establish which node has which index number (rank), and as such the root node sends the rank number to each Client based on its location within the root nodes array of connections.

The number of Server or Client connections a Client node must established can now be computed. Each Client has an array for connections whose size is that of the size of the world. The first entry in the array maintains the connection back to the root node Server Connection). Iterating through all the elements of the array if the index of the array is less than the rank, then a Client connection will be established. If the index of the array is greater than the rank then a Server connection is established.

As each Client can potentially have many Server connections listening for Clients to connect, it is essential to synchronize the establishment of these connections. For example the first step is for Client 1 to establish a Server connection. This Client will then send the index of its current location in the array to the root node, which forwards this on to the corresponding Client (Client 2). Once Client 2 receives the message it can then open a connection to the Server

of Client 1. The address of Client 1 can be found from the array of address transmitted to all the Clients earlier in the process. Just as important as the address to establish the connection to is the channel to which the connection should be made.

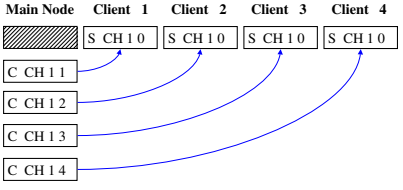


Fig. 1. Initial Connection

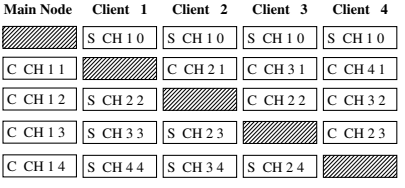


Fig. 2. Complete Interconnect

To know what channel a connection should use to establish communication; it is necessary to know how channel numbers are assigned. All Clients currently maintain Server connections back to the root node these all operate on a particular channel (for example channel 1). As each new Server is established the channel number is incremented. So the second Server will operate on channel 2, the third on channel 3 and so on. The Client that is trying to connect to a Server connection can connect to the proper channel based on the rank of the node and the channel number back to the root node. Figure 2 show this interconnect far more clearly.

The end result is the successful establishment of channels of communication between all nodes within the world. This allows for any Client to communicate with any other Client in a seamless way. Simply by indicating the correct node (index) number one can communicate with any other node in the system. The same as how communication in MPI is carried out.

The entire process outlined in this section is carried out when the MMPI object is constructed. It is however necessary to pass a parameter to indicate if a node is to act as a Server (Root node) or a Client. In the MPI world the establishment of the world is achieved through the Init(...) function / method.

3.3 Communication

In MPI Point to Point communication is carried out using the send(...) , recv(...) functions, as does the MMPI system. The Ping Pong example (Listing 1.1) is identical in structure to that of an MPI Ping Pong program. The sending and receiving of data is carried out on nodes 0 and 1. The sending and receiving data must be both be an Array type Object, for example an array of Strings, Ints, Longs. As with standard MPI programming a send on one node must match up with a receive on the destination node. In the case of MMPI five parameters are necessary for the sending and receiving of data. The parameters being; the input or output buffer, the starting position of the array, the number of elements to send or receive, the data type and finally the id of the node to which to send data or the id of the node from where the data is to be read.

```
mpiNode = new MMPI(nodeType);
rank = mpiNode.getRank();
int[] sendBuff = {rank}; int[] recvBuff= new int[sendBuff.length];
if(rank == 0){
  mpiNode.send(sendBuff,0,1,MMPI.INT,1); mpiNode.recv(recvBuff,0,1,MMPI.INT,1);
  System.out.print("Node 0 Got ["+recvBuff[0]+"]");
}else if(rank == 1){
  mpiNode.recv(recvBuff,0,1,MMPI.INT,0);System.out.print("Node 1 Got ["+recvBuff[0]+"]" ↩
       );
  mpiNode.send(sendBuff,0,1,MMPI.INT,0);
} OUTPUT: Node 0 Got [1] Node 1 Got [0]
```

Listing 1.1. MMPI Ping Pong Example

Parallel programs often require coordinated communication for example an array of data may need to be distributed between all the nodes in the system. This type of global operation can be implemented by using the Send and Receive functions. To simplify this type of communication process a suite of collective communication methods are provided in the form of the Scatter and Gather. Broadcast is also another very useful function that allows for the broadcasting of the same data to all nodes in the system.

3.4 Fault Tolerance

MPI does not have any Fault Tolerance built in to the system. If any node of the world should die then the entire parallel operation will be aborted. Fault Tolerant MPI (FT-MPI) [19] is an implementation to resolve this problem (first released November 2003). FT-MPI is capable of surviving the crash of n-1 processors. If required it can restart the crashed nodes. This however is quite an expensive operation, and the application has to be able to recover the data of the crashed nodes [12]. In the event of a node of the MMPI system crashing then the default operation is to abort the processes of all nodes.

3.5 Running an MMPI Application

As can be seen from the constructor of the MMPI class a parameter is required to indicate whether the application is to run as a Client or a Server. This is necessary only to the MMPI class so that the appropriate BTClient or BTServer objects can be instantiated according to the type of system to run. To allow for this selection of modes all MMPI applications require a simple GUI to allow the user to choose the appropriate mode for the device (Client / Server). Future developments of the system would hopefully have an automatic deployment feature that would rule out the need for user intervention to choose the mode. To get a complete MMPI system running with several nodes it is necessary to start up all the Client nodes first. The Server node will carry out device discovery when initialised and create the world from the active Clients that it detects.

4 Collaborative Computer Graphics

To carry out parallel computer graphics typically requires and extension to most MPI implementation, this extension is called Multi Processing Environment (MPE). It provides tools and utilities for tracing and logging as well as a set of graphical visualisation tools. We now present an example (Mandelbrot Set) where by the MMPI system can be used for the generation of complex images in parallel.

4.1 Collaborative Mandelbrot Generation

The parallel generation of the Mandelbrot Set is a classical application of parallel computation [8] [23]. It is often described as an "embarrassingly parallel computation" as the image can be easily divide into a number of completely independent parts. Each of these parts can be computed on separate processors with out any interdependency on results from other nodes.

The execution of the Parallel Mandelbrot Set application as with any other MMPI program the user must be provided with an interface to choose the type of the application to run (Client or Server). The program has been designed so that the root node provides a Graphical User Interface through which the user can decide on the parameters for the fractal image. The root node issues the request to create the image to all other nodes in the system and will assemble them into one final image for display on screen to the user. This approach mimics MIMD type architecture but the simpler SIMD approach could also have been taken where by the root node would also partake in the image generation process.

```
mpiNode = new MMPI(appType);
while(true){
  mpiNode.recv(inputDataArr, 0, 1, MMPI.STRING, 0);
  parseDataSetVariables(inputDataArr[0]);
  for (int i = 0; i < SIZEX; i++)for (int j = 0; j < SIZEY; j++) {
    Complex c = new Complex( (XMIN + i * STEPX), (YMIN + j * STEPY));
    Complex z = new Complex();
    for (k = 0; k < NR_ITER; k++) {
      z = f(z, c);
      if (z.getAbs() > R) {
        r = c[k % l][0]; g = c[k % l][1]; b = c[k % l][2];
        pixels[ (j * SIZEX) + i] = b + (g << 8) + (r << 16) + alpha; break;
  } } }
  int[] imgSliceArr = {imgSlice}; int[] dataSizeArr = {pixels.length};
  mpiNode.send(imgSliceArr,0,1,MMPI.INT,0);
  mpiNode.send(pixels, 0, pixels.length, MMPI.INT, 0);
}
```

Listing 1.2. Mandelbrot Example Code

The Mandelbrot Generation function is generally very simple (Listing 1.2). One can clearly see that if the absolute value of the complex number lies outside the threshold R, that the pixel at the current coordinates of the iteration through the image will be drawn in a specific colour. The message passing section of code

in the example starts with the construction of a new MMPI object through which all communication is carried out.

Before the Mandelbrot algorithm is executed the parameters must firstly be read in from the root node using the following method mpiNode.recv(...). The input data is received in the form of an array and so it is passed to a method to parse out the data and initialise the variables for the fractal algorithm.

Once the fractal generation algorithm is complete (the result of which is an array of colour data) the pixel data must be returned to the root node. This is carried out by the sending of three messages. The messages are firstly the imageSlice number to indicate what section of the image has been generated. Next the amount of data to receive (so the root node can setup the appropriate buffers). Finally the pixel data is send out.

On the root node once the work has been distributed to all the nodes in the system, the root node must prepare to receive image data from the worker nodes (Listing 1.3). As the worker nodes send three messages the root must so too receive three messages. Once the pixel data has been read a new Image segment is created by calling the createRGBImage method of the Image class passing in the pixels read from the worker node. This continues until all sub images have been read, at which time the completed image will be displayed to the user.

```
mpiNode = new MMPI(appType);
for (int i=0; i < noConnections; i++){mpiNode.send(sectionData[i],0,1,MMPI.STRING,i+1) ↩
    ;}
for (int i=0; i < noConnections; i++){
  mpiNode.recv(segArr,0,1,MMPI.INT,i+1);mpiNode.recv(sizeArr,0,1,MMPI.INT,i+1);
  dataArr = new int[sizeArr[0]];mpiNode.recv(dataArr,0,sizeArr[0],MMPI.INT,i+1);
  fractalImageArray[segArr[0]] = Image.createRGBImage(dataArr,sliceWidth,height,false);
  statusField.setText( "Images Received: " + (i+1) );
} Display.getDisplay(thisMidlet).setCurrent(imgCanvas);
```

Listing 1.3. Root Node Example Code

4.2 Load Balancing

Many methods may be used to decide how to divide up the work between nodes for processing. These include Uniform Block, Cyclic and Dynamic Load Balancing.

Both Uniform Block and Cyclic methods divide the image area into strips on either the horizontal or vertical axis (Figure 3). In the case of Uniform Block the number of strips is equal to the number of nodes that will be used for the processing. So node i receives for computations the image chunk with the consecutive columns $i \times w/nrNodes, i \times w/nrNodes+1, ..., (i+1) \times w/nrNodes-1$. With this geometric division all nodes receive the same amount of image area to process but the amount of processing that each node will have to carry out may vary widely.

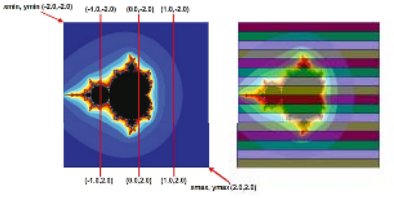


Fig. 3. Uniform and Cyclic Balancing

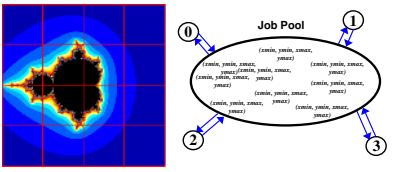


Fig. 4. Dynamic Load Balancing

A better approach but still using the same methodology is Cyclic Load Balancing. The image is still divided into equal sized vertical strips $\{S_0, S_1, ..., S_{p-1}\}$ each of them containing only a few columns. The partition of this p strips onto the nodes is performed into a cyclic matter so that the node i would receive the strips $\{S_{i+j\times nrNodes} : \ j = 0, 1, ..., \frac{p}{nrNodes} - 1\}$.

An alternative to the previous two methods is Dynamic Load Balancing (Figure 4) where by the image area is divided into a regular grid. This grid of work units is formed into a pool of jobs and is distributed among the nodes for processing. Once a node has completed a work unit and returns the results a new work unit will be issued to the node. This process continues until all work units have been completed. The finer the granularity the more even the work load will be between the nodes, however one has increased communication costs.

4.3 Execution Results

A significant amount of time (55,657ms on a Nokia 6630) is required to generate a 200×200 pixel image of the Mandelbrot Set. Both the 6630 and the 6690 use the ARM5 processor at 220Mhz. However the overall speed of the 6630 is marginally better than the 6680 [26]. SPMarkJava06 [14] is a useful tool for testing device speeds.

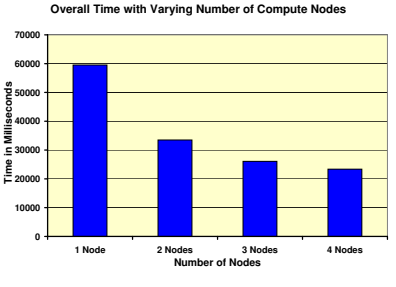


Fig. 5. Total Time Reduction as the Number of Nodes Increases

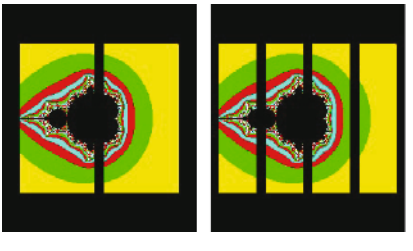


Fig. 6. Example Mandelbrot Images generated with differing World Sizes

Using the MMPI system and distributing the processing out to several nodes can dramatically reduce the computation cost (Table 1, Figure 5, Table 2). The

system was tested using four Nokia 6630's. Figure 6 shows the resultant images for both two and four compute nodes.

Table 1. Overall Times with Varying Number of Compute Nodes

4 Nodes	3 Nodes	2 Nodes	1 Node
23,362ms	26,074ms	33,488ms	59,528ms

Table 2. Image Generation Times for the Mandelbrot Set, 500 iterations

Node 1	Node 2	Node 3	Node 4
11,627ms	19,462ms	17,228ms	6,512ms

5 Collaborative Games

The MMPI library has huge potential in the area of collaborative gaming over a Bluetooth network. One such example [17] shows that by combining Bluetooth and the ARToolkit an interactive game of tennis can be produced between two players. Having a library to abstract from the inner workings of Bluetooth would greatly simplify the development cycle of such games. Using the library the game could be expanded to four players (to allow for a doubles match).

Another example of collaborative gaming is the classical Pacman game, updated to allow for multiple players at the same time would add another level of interest and playability. Fantasy games where players have certain points levels, stamina, hit points are also well suited to Bluetooth as the amount of data to be passed between the engaged devices is very small (generally just status values of a players condition) [20]. Strategy games are of course a great area of potential for this system. A single user playing against current Artificial Intelligence players is one thing, but to vie for the winning position against a human player is a different matter entirely, where the opponents actions cannot be predicted.

6 Conclusion

An overview of the MMPI library has been given. It has been shown that the library has many uses in the area of mobile computing. It provides a simple to use interface for the software developer to develop parallel applications using Bluetooth as the communications medium. Its uses are not just limited to the area of scientific computing but also provides a useful base for collaborative gaming across multiple mobile devices. One can also use to library to facilitate interactive student/teacher M-Learning within a classroom environment. In general the library is a useful tool for anywhere mobile Bluetooth communication is required.

References

1. ARM, ARM Cortex-A8, *http://www.arm.com/products/CPUs/ARM_Cortex-A8.html* .
2. ARM, ARM Introduces Industry's Fastest Processor For Low-Power Mobile And Consumer Applications, *http://www.arm.com/news/10548.html* , Oct, 2005.

3. M. Baker, D. Carpenter, MPJ: A Proposed Java Message-Passing API and Environment for High Performance Computing, *In Proceedings of IEEE International Parallel & Distributed Processing Symposium*, 2000.
4. Berkeley, Berkeley Open Infrastructure for Network Computing (BOINC), *http://boinc.berkeley.edu/*.
5. Berkeley, Search for Extraterrestial Intelligence (SETI), *http://setiathome.berkeley.edu/* .
6. Bluetooth.com, The Official Bluetooth Website, *http://www.bluetooth.com/*.
7. Bluetooth.org, The Official Bluetooth Membership Site, *http://www.bluetooth.org/*.
8. M. Book, Parallel Fractal Image Generation, *http://www.mattiasbook.de/papers/parallelfractals/*.
9. Distributed Systems Group (Portsmouth), Message Passing in Java (MPJ) Project, *http://dsg.port.ac.uk/projects/mpj/*.
10. D. Doolan, S. Tabirca, Distributed Fractal Generation Across a Piconet, *In Proceedings of SIGRAD 05, Annual SIGRAD Conference, Special Theme: Mobile Graphics*, Lund, Sweden, 2005, 63–68.
11. D. Doolan, S. Tabirca, Interactive Teaching Tool to Visualize Fractals on Mobile Devices, *In Proceedings of Eurographics Ireland Chapter Workshop*, 2005, 7–12.
12. G. Fagg et. al, Scalable Fault Tolerant MPI: Extending the Recovery Algorithm, *In Proceedings of* 12^{th} *European Parallel Virtual Machine and Message Passing Interface Conference - Euro PVM/MPI* Sept, 2005.
13. Finfacts, Worldwide Mobile Phone Sales Increase 22 Percent in the Third Quater of 2005, *http://www.finfacts.com/irelandbusinessnews/publish/article_10004020.shtml* , Nov, 2005.
14. Futuremark, SPMarkJava06 Benchmark for Mobile Devices, *http://www.futuremark.com/products/spmarkjava06/* .
15. P. Greenspun, Mobile Phone as Home Computer, *http://philip.greenspun.com/business/mobile-phone-as-home-computer* , Sept, 2005.
16. E. Hardy, Bluetooth Kicking into High Speed, *http://www.brighthand.com/article/Faster_Bluetooth_Standard_Proposed* 2004.
17. A. Henrysson, M. Billinghurst, M. Ollila, Augmented Reality of Mobile Phones - Experiments and Applications, *In Proceedings of SIGRAD 05, Annual SIGRAD Conference, Special Theme: Mobile Graphics*, Lund, Sweden, 2005, 35 – 40.
18. HPJava Project, mpiJava, *http://www.hpjava.org/mpiJava.html* .
19. ICL, Fault Tolerant MPI, *http://icl.cs.utk.edu/ftmpi/index.html* .
20. B. Long, A Study of Java Games in Bluetooth Wireless Networks, *Masters Thesis* University College Cork, 2004.
21. MPI, The Message Passing Interface (MPI) Standard, *http://www-unix.mcs.anl.gov/mpi/*.
22. MPICH, MPICH - Free Implementation of MPI, *http://www-unix.mcs.anl.gov/mpi/mpich/*.
23. C. O'Mahony, Distributed Multimedia Processing, *Masters Thesis*, University College Cork, 2004.
24. F. Pilato, ARM Reveals 1Ghz Mobile Phones Processors , *http://www.mobilemag.com/content/100/102/C4788/*, Oct, 2005.
25. Sun Microsystems, Java 2 Platform, Micro Edition (J2ME), *http://java.sun.*com/j2me/index.jsp.
26. Symbian Freak, Nokia 6680 is Loosing the Battle to 6630, *http://www.*symbian-freak.com/news/0305/6680.htm.

Study on Application Server Aging Prediction Based on Wavelet Network with Hybrid Genetic Algorithm*

Meng Hai Ning[1,**], Qi Yong[1], Hou Di[1], Liu Liang[2], and He Hui[1]

[1] School of Electronics and Information Engineering, Xi'an Jiaotong University, Xi'an, China, 710049
mengning2001ji@163.com
[2] IBM China Research Laboratory, Beijing, China

Abstract. Software aging is an important factor that affects the software reliability. According to the characteristic of performance parameters of application sever middleware, a new model for software aging prediction based on wavelet networks is proposed. The structure and parameters of wavelet network are optimized by hybridization of genetic algorithm and simulated annealing algorithm. The objective is to observe and model the existing resource usage time series of application server middleware to predict accurately future unknown resource usage value. Judging by the model, we can get the aging threshold before application server fails and rejuvenate the application server before systematic parameter value reaches the threshold. The experiments are carried out to validate the efficiency of the proposed model, and show that the aging prediction model based on wavelet network with hybrid genetic algorithm is superior to the neural network model and wavelet network model in the aspects of convergence rate and prediction precision.

1 Introduction

Recent studies have reported the phenomenon of software aging [1, 2] in which the state of system performance degrades with time. The primary symptoms of this degradation include exhaustion of system resources, data corruption and instantaneous error accumulation. This may eventually lead to performance degradation, crash/hang failure, or other unexpected effects. Aging has not only been observed in software used on a mass scale but also in specialized software used in high-availability and safety-critical applications [1].

In order to enhance system reliability and dependability and prevent degradation or crashes, a preventive maintenance technique called software rejuvenation was introduced [1]. This involves occasionally stopping the running software, cleaning its internal state and then restart. For optimizing the timing of such a preventive maintenance, it is important to detect software aging and predict the time when the resource exhaustion reaches the critical level. Most of the previous measurement techniques

* This project is supported by the National Natural Science Foundation of China under Grant No. 60473098 and IBM CRL Joint Project.

** Biography: Meng Hai Ning(1979-), female, Ph.D candidate, research direction: distributed computing, middleware system, middleware reliability. E-mail:mengning2001ji@163.com.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 573 – 583, 2006.
© Springer-Verlag Berlin Heidelberg 2006

for dependability evaluation were based on data from failure events [3, 4]. Estimation of the failure rate and mean time to failure of widely distributed software was presented in [3]. The approach for failure prediction was described in [5], which is based on an increase in observed error rate, an error number threshold, a CPU utilization threshold or a combination of the above factors. For the reason that software aging cannot be detected or estimated via collecting data at failure events only, by contrast, periodically monitoring and recording of the activity system parameters in operation is adopted in our works. The data related to system parameters are extracted from application server at regular intervals, therefore, the extracted data can be considered as the time series of system parameters.

So far, many kinds of methods for time series prediction have been proposed, such as neural network [6], wavelet network [7-9], wavelet transform [10, 11], Bayesian theory [12] and support vector machine [13]. Neural networks are powerful tools for fitting nonlinear time series. However, the implementation of neural networks has disadvantages in determining the parameters of neurons and constructing network structure. Furthermore, as the training processes of neural network often settle in undesirable local minimal of the error surface, the network convergence rate is slow. Wavelet transform [10, 11] is a useful tool for multi-resolution decomposition of time series, so that slight temporal structures and the trend of time series can be revealed. Wavelet networks [8] can make up for the deficiencies of both wavelet and neural network and construct network topology efficiently. The key problem is to design an algorithm to determine the number of hidden nodes and train the network to adjust the parameters of the network to minimize the cost function, which usually is the square error between the output of the network and the actual. The usual method to do this in the domain of neural networks is based on the BP algorithm. Since genetic algorithm [14] is an optimization method, it can be adopted in wavelet network to help search the optimum number of hidden nodes and parameters of neural network and overcome the problem of convergence towards local optima.

In this paper, wavelet network method with genetic algorithm and simulated annealing algorithm is proposed to predict resource usage for the purpose of detecting aging in application sever. Firstly, the collected operating system resource usage and system activity data at regular intervals are decomposed by wavelet. The decomposed coefficients are considered as the inputs of Wavelet network. Then the training of wavelet network is optimized by hybrid genetic algorithm to search the optimum number of hidden nodes and parameters of wavelet network. The prediction data of resource usage, which is the output of wavelet network, are then reconstructed by inverse wavelet transformation. Eventually, the experimental results are demonstrated to validate the efficiency of the proposed method, and show that the aging prediction model based on wavelet network with hybrid genetic algorithm is superior to the BP neural networks model and wavelet network model [8] in the aspects of convergence rate and prediction precision.

The main contributions of this paper are as follows:

1. Propose a wavelet- network based aging prediction model.
2. Combine genetic algorithm and simulated annealing algorithm to optimize the structure and parameters of wavelet network.
3. Use the resource parameters data collected from JUFrame application server system to evaluate the aging prediction performance.

2 Wavelet Transform and Wavelet Network

2.1 Wavelet and Wavelet Transforms

We briefly recall some basic concepts about wavelet transforms that will be useful for developing wavelet network.

Definition 1. Let $\psi(x) \in L^2(R)$ be a mother wavelet. The corresponding family of dilated and translated wavelet is defined by

$$\psi_{a,b}(x) = a^{-\frac{1}{2}} \psi(\frac{x-b}{a}) \quad a,b \in R, a > 0 . \tag{1}$$

Definition 2. If a and b are properly selected, the wavelet $\psi_{a,b}(x)$ can constitute a frame of $L^2(R)$ expressed as follows:

$$A \| f \|^2 \leq \sum_a \sum_b | \langle f, \psi_{a,b} \rangle |^2 \leq B \| f \|^2 . \tag{2}$$

where $f \in L^2(R)$, $A > 0$ and $B > 0$ are the frame bounds. If A=B, $\{\psi_{a,b}\}$ is the tight frame. IF A=B=1, $\{\psi_{a,b}\}$ is an orthogonal basis.

Definition 3. A function $f(x)$ in $L^2(R)$ space can be analyzed as follows

$$f(x) = \sum_{a,b}^{N} \langle f, \psi_{a,b} \rangle \psi_{a,b} = \sum_{a,b}^{N} C_{a,b} \psi_{a,b} . \tag{3}$$

Definition 4. The continuous wavelet transform (CWT) is defined as follows:

$$(W_\psi f)(a, b) = < f, \psi_{a,b} > = | a |^{-1/2} \int_{-\infty}^{+\infty} f(t) \psi(\frac{t-b}{a}) dt . \tag{4}$$

Wavelet transform decomposes the function $f(x)$ into a wavelet function series and uses such a series to approximate the function $f(x)$.

2.2 Wavelet Network

Wavelet network are based on the continuous wavelet, orthogonal wavelet and wavelet frame, and can be obtained by substituting a wavelet function for the activation function of the hidden nodes. According to the wavelet theory, the wavelet networks have the universal approximation ability in L^2 space. Mathematical formula for the wavelet networks were provided in [8]. The wavelet network structure is illustrated in Fig.1, and the output of wavelet network $g(x)$ can be expressed as follow:

$$g(x) = \sum_{i=1}^{n} w_i \psi[D_i R_i (x - t_i)] + \overline{g} . \tag{5}$$

where w_i are adjustable weights of the connections, t_i are translation vectors, D_i are diagonal dilation matrices, R_i are rotation matrices, N is the number of hidden nodes (wavelons) in the hidden layer and the additional parameters g is introduced to help dealing with nonzero mean functions on finite domains, because the wavelet $\psi(x)$ has zero mean.

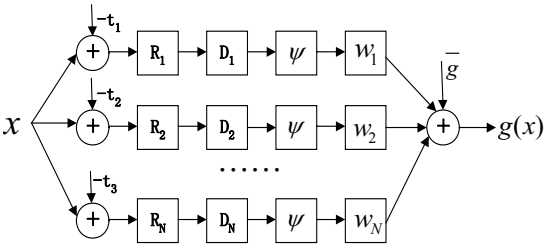


Fig. 1. Wavelet network structure

Two key problems in designing of wavelet network are how to determine the structure and parameters of wavelet network, and what learning algorithm can be effectively used for training the wavelet network.

3 Software Aging Prediction Model

3.1 Wavelet Analysis Technology

The time series data of memory usage has the characteristic of non-linear, strong-correlated, multi-fractal and chaos. If the fluctuation rule of memory usage time series can be distinguished and analyzed, the trend of memory usage can be predicted. Wavelet transformation is used to enhance the efficiency of the optimization process.

The wavelet transformation formula (4) is a direct numerical integral calculation of the wavelet coefficients dot by dot in the spatial domain of memory usage time series. However, it is time-consuming. Mallat algorithm in [15] is adopted to decompose and reconstruct the memory usage time series. The original time series is decomposed into detail coefficients and approximation coefficients, in which detail coefficients can be used for "detail" analysis and prediction, and approximation coefficients can be used for the analysis and prediction of the slow trends in the time series.

3.2 Wavelet Network Schema

The wavelet's coefficients of each scale are input into the wavelet network for prediction. Fig.2 illustrates the design schema of wavelet network.

The wavelet network includes three layers:

Layer 1 includes n input variables $x_1, x_2, \cdots, x_n$;

Layer 2 is a hidden layer that consists of wavelet function substituting for activation function. Weight $w1$ links the input nodes and the hidden nodes. Wavelet function is expressed as follows:

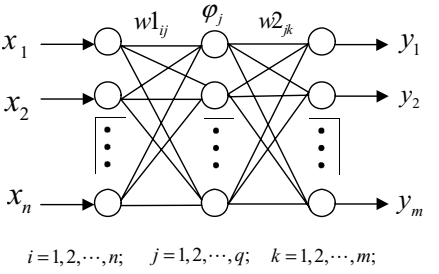


Fig. 2. Wavelet network schema

$$\varphi_j(x) = s_j^{-\frac{1}{2}}\psi(\frac{x-t_j}{s_j}) \quad (j=1,2,\cdots,q). \tag{6}$$

where s_j, t_j are dilation and translation factors of mother wavelet ψ. φ is a set of daughter wavelets generated by dilation s and translation t from a mother wavelet ψ. In this paper, Morlet wavelet is selected as a mother wavelet expressed as follows:

$$\psi(x) = \cos(1.75x)e^{-\frac{x^2}{2}}. \tag{7}$$

By substitution (6) to (7), the following formula is obtained.

$$\varphi_j(x) = s_j^{-\frac{1}{2}}\cos\left(1.75\cdot\left(\frac{x-t_j}{s_j}\right)\right)e^{-\frac{\left(\frac{x-t_j}{s_j}\right)^2}{2}}. \tag{8}$$

Layer 3 is an output layer that sums the production of output value of the hidden nodes and the output connection weight $w2$ that is between the hidden nodes and the output nodes.

$$y_k = \sum_{j=1}^{q} w2_{jk}\varphi_j \quad (k=1,2,\cdots,m). \tag{9}$$

From the theory above, the wavelet network formula can be deduced as follows

$$y_k(x) = \sum_{j=1}^{q} w2_{jk}\psi\left(\frac{\sum_{i=1}^{n} w1_{ij}x_i - t_j}{s_j}\right) \quad (k=1,2,\cdots,m). \tag{10}$$

3.3 Learning Algorithm of Wavelet Network Based on Hybrid Genetic Algorithm

It is difficult to decide the best structure of the wavelet network. Due to the fact that the learning algorithm of wavelet network often settles in undesirable local minima

and converges slowly, genetic algorithm is adopted here to help search the optimum number of hidden nodes and parameters of neural network such as the number of hidden nodes, the parameters of dilation and translation, and the connection weights. On the other hand, genetic algorithm is an optimization method mainly based on the concepts of natural selection and evolutionary process. However, the convergence is slow for the reason that the control probabilities for operations such as crossover and mutation are usually constant during the optimization process. Simulated annealing algorithm is combined with genetic algorithm to smooth the convergence process.

3.3.1 Genetic Operator

Chromosome Encoding. A chromosome consists of three parts shown in Fig.3. Parameters of wavelet network are decimal coded and number of hidden nodes is coded in binary string. The bit of '1' in the binary string indicates the corresponding hidden node is valid, and the bit of '0' in the binary string indicates that is invalid. l is number of hidden nodes, and the maximum value of l is selected by experience. Each connection weights wl_{ij} , $w2_{jk}$, input x_i , translation factor t_i and dilation factor s_i are set from the domain (-1 1) using random generator.

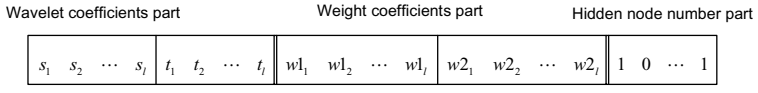


Fig. 3. Chromosome encoding of wavelet network for aging prediction

Fitness evaluation. The least-squared error function e is used to represent the unfitness value of the genetic wavelet network associated with one individual. Thus, the fitness function f is defined as follows:

$$f = \frac{1}{e} = \frac{1}{\frac{1}{2}\sum_{l=1}^{p}\sum_{k=1}^{m}(d_{lk} - y_{lk})^2} . \tag{11}$$

where y_{lk} is the computing value of the lth sample on the kth output node in the wavelet network. And d_{lk} is the actual value of the kth output node accordingly.

Crossover operator. The crossover operation is a process by which new offspring generated from parent during reproduction. The gene in chromosome of two parents is across selected one by one as the gene of the offspring with probability of p_c.

Mutation operator. Mutation operator plays an important role in introducing new gene to the chromosomes with probability of 1-p_c. And the mutation operator diversifies the search and prevents the premature convergence that leads to nearly the same individuals within a population after several generations. The mutation probability must be sufficiently small to ensure that the crossover is the primary means of creating new offspring.

3.3.2 Hybrid Genetic Algorithm for Training Wavelet Network

The main steps of training wavelet network with hybrid genetic algorithm and simulated annealing algorithm can be summarized as the following algorithm 1.

Algorithm 1. Hybrid genetic algorithm for training Wavelet network

1. Input wavelet coefficients of memory usage time series;
2. Generate chromosome coding of initial population G(0) at random, initiate the temperature of wavelet network t(0) and set i=0;
3. REPEAT
   1) Compute the fitness value of each individual in the population according to neural network BP learning method;
   2) Use simulated annealing (SA) algorithm for each individual of G(i) as follows:
      i. Generate new individual by using state of SA;
      ii. Calculate $\triangle C$, the difference of fitness value between the new individual and the old one;
      iii. Calculate $Pr=\min[1,\exp(-\triangle C/t(i))]$ as the accepted probability;
      iv. If Pr > random [0,1], then the new individual substitutes for the old one, so that wavelet network with a high temperature are mutated severely and that with a low temperature are mutated only slightly;
   3) Sort the individual of the population in descent order according to the corresponding fitness value;
   4) Set n is the number of individuals eliminated from the population G(i), and set j=0;
   5) The new generation G(i+1) evolves from the population G(i) by genetic operator, and the genetic operation procedure is as follows:
      WHILE j < n
      i. Randomly select two parents with higher fitness value from top n individuals of the sorted population;
      ii. Apply crossover operation or mutation operation to generate the offspring of the two selected parents with probability of Pc and 1-Pc separately, and the new offspring replace the n-j individual of the sorted population G(i);
      iii. j++;
   6) Set $t(i+1)=v\cdot t(i)$, where $v\in(0,1)$ is the rate of annealing temperature;
4. UNTIL termination criterion is satisfied or generation number reaches the given maximum generation number.

4 Experimental Results and Discussions

4.1 Experimental Setup and Data Collection

The experimental environment consists of a JUFrame application server, clients and database server. In the client, the load generator is used to generate requests to the application server. JUFrame application server connects and queries database server, and then returns results to clients. By load generator model and resource monitor model, the dynamic parameters in clients and application server are periodically monitored and recorded in a certain format separately. The experimental setup schema is presented in Fig.4.

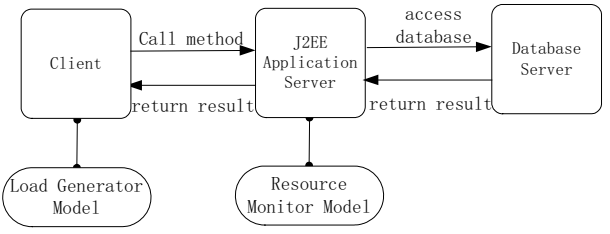


Fig. 4. Experimental setup schema

The sampling interval of resource usage data is twelve minutes. System parameter of memory usage for 100 hours is extracted from recorded file to predict aging of application server.

4.2 Prediction of Memory Usage Time Series

Time series of memory usage is illustrated in Fig.5. At first, the original data requires normalizing, a process of standardizing the possible numerical range that the input variables can take. The procedure involves finding the maximum and minimum elements and then normalizing the input vectors x_i to the range [0, 1].

$$\hat{x}_i = \frac{x_i - \min}{\max - \min}. \tag{12}$$

The normalized data are decomposed by wavelet transformation. Thus the memory usage time series is transformed in three stages from spatial domain to frequency domain. The behavior of the three-stage decomposed wavelet coefficients for memory usage time series is shown in Fig.6, from which the conclusion can be drawn that as the decomposition level increases the corresponding coefficients become smoother. The approximation coefficients show the trend of memory usage.

Normalized mean square error (NMSE) is adopted as indicator of performance evaluation for aging prediction. NMSE is defined as follow:

$$NMSE = \frac{1}{\sigma^2 n}\sum_{k=1}^{n}(x(k) - \hat{x}(k))^2 . \tag{13}$$

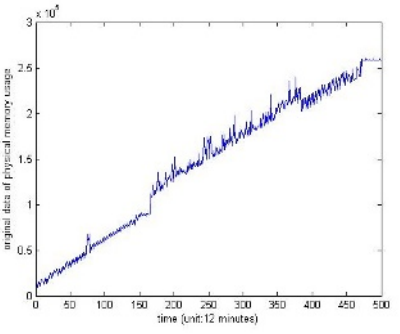


Fig. 5. Original data of memory usage

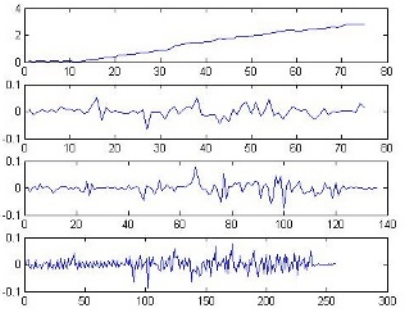


Fig. 6. Illustration of the wavelet decomposition of the memory usage time series. From top to bottom: the decomposition approximation coefficient in stage 3, the detail coefficient in stage 3, stage 2 and stage 1.

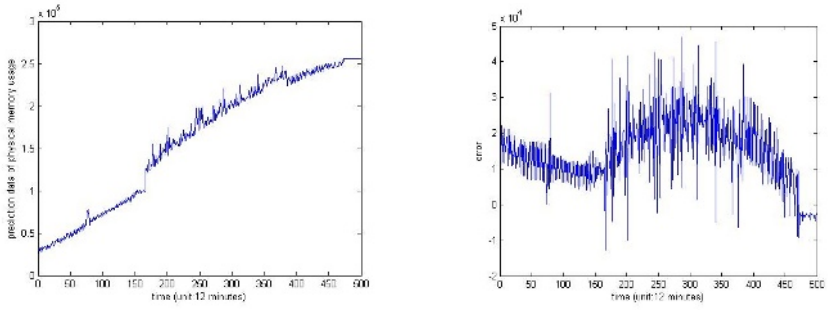
where $x(k)$ is the actual value of the time series, $\hat{x}(k)$ is the forecasting value, n enumerates the points of training data set, and σ^2 is the variance of the actual value of time series over the forecasting period.

The wavelet function is taken as Morlet wavelet. The maximal number of the hidden nodes is set to 80. The learning rate is set to 0.01. The maximum generation is determined as 1200. The number of population is set to 50.

Table 1 presents approximation performance based on wavelet network model compared with neural network and wavelet network model. The table is shown that prediction precision of wavelet network with hybrid genetic algorithm is superior to that of neural network and wavelet network. Fig.7 displays the prediction data for one-step forward prediction model of memory usage and the error between original data and prediction data. From the figure, we can see that wavelet network model based on hybrid genetic algorithm is effective and it can predict memory usage time series with lower error. We can also see system performance decreases with time. Fig.8 illustrates the generation number and corresponding maximum fitness value. As is shown, when generation number reaches to 590, the fitness value convergences to 38.22 and the maximum fitness has been achieved.

Table 1. Comparison of approximation performance

Models	Number of hidden nodes	NMSE
Wavelet network with hybrid genetic algorithm	26	0.0261
Wavelet network in reference [8]	42	0.0363
Neural network in reference [8]	60	0.0573



(a) (b)

Fig. 7. Prediction data of memory usage for one step forward prediction (a). Prediction data, (b) Error between original data and prediction data.

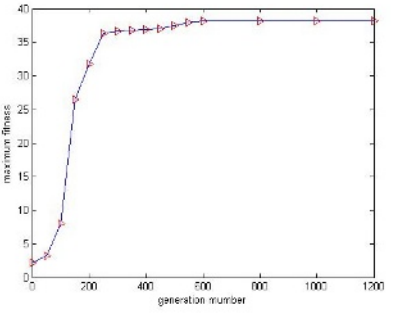


Fig. 8. Relation between generation and fitness in hybrid genetic algorithm training wavelet network

5 Conclusions

The effectiveness of wavelet network with hybrid genetic algorithm for aging prediction has been investigated. The original time series is decomposed into wavelets. Then the decomposed coefficients are predicted by means of wavelet network, and an algorithm of back-propagation with genetic algorithm and simulated annealing is proposed for wavelet network learning. Through adopting multi-encoding, this algorithm can optimize the structure and the parameters of wavelet network in the same training process, therefore, the structure of wavelet network can be more reasonable

and the local minimum problem in the training process will be overcome efficiently. Compared with previous work on wavelet network and neural network to time series prediction, the method proposed in this paper has superiority in aspects of convergence rate and prediction precision.

It is important to predict the critical resource usage such as memory usage for application server. Thus software aging can be detected and the aging threshold before server crashed can be evaluated using the prediction model. Future work includes the aging prediction model considered more causations of resource resumption.

References

1. Garg, S., Puliafito, A., Telek, M., Trivedi, K.S.: A Methodology for Detection and Estimation of Software Aging. Int. Symp. On Software Reliability Engineering, ISSRE (1998)
2. Huang, Y., Kintala, C., Kolettis, N., Fulton. N: Software Rejuvenation: Analysis, Module and Applications. IEEE Int. Symposium on Fault Tolerant Computing, FTCS 25 (1995).
3. Chillarege, R., Biyani, S., Rosenthal, J.: Measurement of failure rate in widely distributed software. In Proc. of 25th IEEE Intl. Symposium on Fault-Tolerant Computing, Pasadena, CA (1995) 424-433
4. Tang, D., Iyer, R. K.: Dependability measurement modeling of a multicomputer system. IEEE Transactions on Computers, vol. 31. (1993)
5. Lin T-T., Siewiorek, D. P.: Error log analysis: Statistical modeling and heuristic trend analysis. IEEE Transactions on Reliability, vol. 39. (1990) 419-432
6. Amir, B., Geva: Scalenet-Multiscale Neural network Architecture For Time Series Prediction. IEEE Transactions on Neural Networks, Vol. 9. (1998) 1471-1482
7. Zhang, Q., Benvenise, A.: Wavelet network. IEEE Transactions on Neural Network, vol. 3. (1992) 889-898
8. Sheng-Tun Li, Shu-Ching Chen: Function Approximation Using Robust Wavelet Neural Networks. The 14th IEEE International Conference on Tools with Artificial Intelligence, Taiwan (2002) 483-488
9. Bashir, Z., El-Hawary M.E: Short Term Load Forecasting By Using Wavelet Neural Networks. The IEEE Conference on Electrical and Computer Engineering, Canadian (2000) 163 – 166
10. Soltani, S., Boichu, D., Simard, P., Canu, S.: The long-term memory prediction by multiscale decomposition. Signal Processing. Vol. 80. (2000) 2195-2205
11. Bai-Ling Zhang, Coggins, R.: Multi-resolution Forecasting for Future Trading using Wavelet Decompositions. IEEE Transactions on Neural Network, Vol. 12. (2001)
12. Chris, C., Holmes Bani, Mallick, K.: Bayesian Wavelet Networks for Nonparametric Regression. IEEE transactions on neural networks, vol. 11. (2000)
13. Zhang, X.: Robust Multiwavelets Support Vector Regression Network. International Conference on Control and Automation, Budapest,Hungary (2005) 27-29
14. Alei Prochizka, Vladimir, S.: Time Series Prediction Using Genetically Trained Wavelet Networks. Sys, Vladimir Source: Neural Networks for Signal Processing. Proceedings of the IEEE Workshop (1994) 195-203
15. Rioul, O., Duhamel, P.: Fast algorithms for discrete and continuous wavelet transform. IEEE Transactions on Information Theory, vol. 38, (1992) 569-586

Autonomic Service Reconfiguration in a Ubiquitous Computing Environment*

Yoonhee Kim and Eun-kyung Kim

A Dept. of Computer Science, Sookmyung Women's University
Seoul, Korea
{yulan, kimek}@sookmyung.ac.kr

Abstract. A middleware in ubiquitous computing environment (UbiComp) is required to support seamless on-demand services over diverse resource situations in order to meet various user requirements [1]. Since UbiComp applications need situation-aware middleware services in this environment. In this paper, we propose a semantic middleware architecture to support dynamic software component reconfiguration based on ontology to provide fault-tolerance in a ubiquitous computing environment. Our middleware includes autonomic[9] management to detect faults, to analyze causes of them, and to plan semantically meaningful strategies to recover from the failure using pre-defined fault and service ontology trees. We implemented a referenced prototype, Web-service based Application Execution Environment (Wapee), as a proof-of-concept, and showed the efficiency in runtime recovery.

1 Introduction

The advent of Ubiquitous Computing (UbiComp), which runs dynamically over heterogeneous environment, emphasizes the needs of service-oriented middleware services in the concept of anytime, anywhere, and any device computing. In the UbiComp environment, the concept of situation-aware middleware has played an important role in meeting user needs with available computing resources appropriately in dynamic environment. An UbiComp system consists of a heterogeneous set of computing devices; a set of supported tasks; and some infrastructures the devices may rely on in order to carry out their tasks. It hides the heterogeneity of the resource environments and provides necessary services to UbiComp applications.

As the diversity and complexity of situations in UbiComp environment, it is not trivial and realistic to come up with semantically meaningful middleware services to support high availability, especially to recover from faulty situations with predefined recovery strategies in real world. In addition, pursing sophisticated controls over complicated faulty situation takes quite amount of time to analyze the cause and plan recovery strategies, in order to achieve service continuity in various running environment.

Fault-tolerance issues have been addressed in various areas of computing systems such as computer architecture, operating systems, distributed systems, mobile

* This research was supported by the Sookmyung Women's University Research Grants 2005.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 584 – 593, 2006.
© Springer-Verlag Berlin Heidelberg 2006

computing and computer networks. In this paper, we discuss semantically meaningful fault-tolerant middleware architecture to improve availability of application services in UbiComp environments. In this paper, we suggest a semantic middleware architecture to support dynamic software component reconfiguration based fault and service ontology to provide fault-tolerance in a ubiquitous computing environment. To enable a service to seamlessly run in ubiquitous environment, we introduce the Web-service based Application Execution Environment (Wapee). It consists with Fault Management (FM) and Runtime Service Management (RSM) with high fault-tolerance, or continuous availability. The FM provides ontology-based context understanding service in the application areas. The RSM can be dynamically service reconfiguration. Both are presented for the fast execution time, fault-tolerance and continuous availability.

The rest of paper is organized as follows. The related works are introduced in section 2. Section 3 presents overall architecture and the detailed description of Wapee. In section 4, the experiments of our prototype have demonstrated the semantically meaningful fault detection and recovery functionality of the mechanism in our architecture and the efficiency in runtime. We conclude with some directions for future work at the end of this paper.

2 Related Works

Research on fault tolerance has been more emphasized to provide seamless and continuous services in Grid [2], ubiquitous, or distributed computing environment. Grid Enactor and Management Service (GEMS) [3] supports the detection of individual job process failures for parallel message-passing applications. Failed Jobs can be canceled and restarted, either on the same local resource if sufficient nodes are available in a restart queue, or on another resource. GEMS requires that a local resource manager support certain fault-detection and reporting capabilities. CORBA [4] have long lacked real support for fault tolerance. In most cases, a failure was simply reported to the client and the system undertook no further action. For example, if a referenced object could not be reached because its associated server was unavailable, a client was left on its own. In CORBA version 2.6, fault tolerance is explicitly addressed.

The Adaptive Reconfigurable Mobile Objects of Reliability (Armor) [5] middleware architecture offers a scalable low-overhead way to provide high-dependability services to applications. It uses coordinated multithreaded processes to manage redundant resources across interconnected nodes, detect errors in user applications and infrastructural components, and provide failure recovery. The authors describe their experiences and lessons learned in deploying Armor in several diverse fields.

3 Wapee Overview

Wapee (Web-service based Application Execution Environment) is a middleware for UbiComp environments contrived with the aim of supporting an application to

configure and adapt itself to the underlying environments. A key in this architecture is providing a uniform access interface to users over heterogeneous resources and dynamic changes of them. It is not easy to choose most relevant service instances with right context for application's current situation. Service instances are evaluated based on the extent of fitness to current context such as current location, or preferences. To evaluate service fitness, Wapee focuses on providing autonomic fault-tolerance services with fault detection, fault analysis and recovery (see Fig. 1). Application level service reconfiguration can be achieved by autonomic detection and analysis services in application-level Fault Management with semantically meaningful ontology of U-services and faults in a ubiquitous environment. The service reconfiguration information in an Application Description Graph (ADG) is fed in to Runtime Service Management (RSM) to be realized as U-services on a prepared resource pool. Based on the ADG, the RSM asks Autonomic Management Generation (AMD) Service to create Application Deployment Description (ADD), which includes service deployment information such as resource description of service managers, local schedulers, input and output data file path, and executables; and runtime dependency of the U-services in the ADG.

3.1 Runtime Service Management (RSM)

RSM is responsible for instantiating and monitoring service (See Fig. 2). The RSM makes estimates of the resource usage of job submissions in order to ensure efficient use of grid resources [8]. Examples of service failures include service crashes due to bugs and operating system errors, faulty operation of services like sensing incorrect context, wrong inferring delivery of events. Service failures can potentially lead to failure of the UbiComp system.

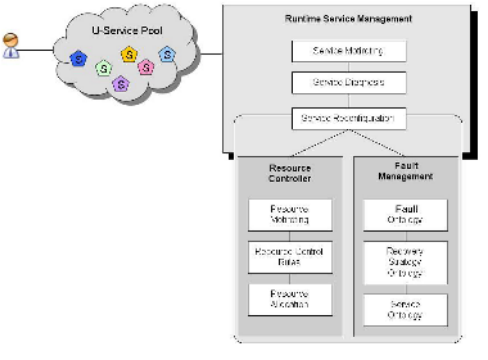


Fig. 1. Overview of Wapee

The purpose of Monitoring service is to provide real-time job monitoring and status feedback to a steering service while operating in close interaction with an execution service, such as Condor [10], to provide interactivity, fault tolerance and error detection. Once a job is submitted in Wapee, Monitoring services periodically

monitors a job that has been submitted for execution in the Virtual Organization (VO) and reports job status. Whenever the state of a job changes the Monitoring service will update the repository. It supports querying job status and monitoring of output and error streams of running jobs. Resource Monitoring Service gathers information of resource in VO [8].

The RSM also addresses autonomic reconfiguration because different invocations of the same service may result in the selection of different components. In the Wapee architecture, it is primarily responsible for planning and initiating configuration changes in the system. Development of this adaptive reconfiguration mechanism requires identification of output information provided by the system and input information that the mechanism can inject into the system to affect change. The dynamic resource management service we have designed is in charge of detecting configuration changes, updating the distribution of directory entries on cluster nodes in the event of a configuration change, triggering reconfiguration of distributed services when needed.

Autonomic Service Reconfiguration interacts with other components of RSM or Fault Management to search currently available services to be suitable to the context change [11]. To adopt new service, it should check and verify available resources or resource conflicts among services to avoid service crush or malfunction of applications.

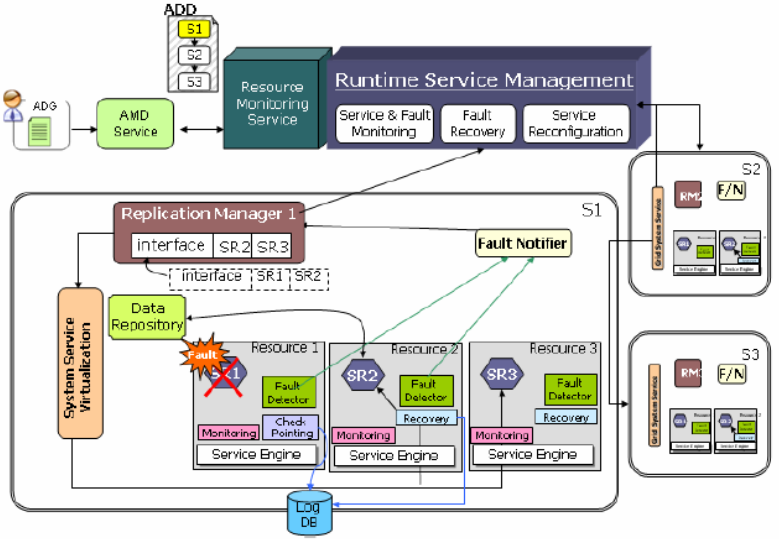


Fig. 2. The architecture of RSM

To meet the requirement of high availability and fault tolerance, replication scheme is used. Fig. 3 depicts the implementation of the Replication Manager (RM) in a typical deployment scenario at a local site replicates data from one or more remote sites.

The RM replicates the data and the processes of an application, and distributes the replicas across the processors in the system. The Fault Detector of Wapee offers their fault notifications to the replication manager, thereby allowing it to restore the degree of replication if a replica has crashed. When a fault occurs, Fault Detectors detect fault in the objects, and report faults to the Fault Notifier. The Fault Notifier receives reports from the Fault Detectors, and propagates the reports as fault event notifications. The Runtime Service Management reasons about the fault reports that it has received. The operations of RM include location, identifying where desired data files exist on the Grid; transfer, moving the desired data files to the local system efficiently; and registration. We considered primary-backup replication for achieving fault-tolerance.

Service Reconfiguration Approaches. Our middleware makes context aware applications easy to be developed and deployed. When context change from an application is acquired and the change is resulted as a fault, the middleware reconfigures Application Deployment Description (ADD) to meet requirements with the help of Fault Management. The Runtime Service Management (RSM) has a reconfigured ADD. To create reconfigured ADD, RSM requests Fault Management (FM) to identify faults and provide recovery strategy using fault, service, and recovery strategies ontology trees. As FM evaluates the situations and identifies a set of possible faults, the service broker retrieves relevant recovery strategies to resolve the faults. Using a semantic relaxation method, a set of extended candidate services is chosen from service ontology. Only the services that can contribute to resolve the fault are selected as candidate services that can b used to substitute the original service. The reconfigured services from FM are notified to RSM to generate ADD. When the reconfigured context in ADD is activated, user level application functionalities will be provided continuously. The following procedure is Autonomic management for fault-tolerance.

User can create the ADG through setting of application and domain. And then ADD is configured and service is initiated. When a fault occurs during execution, an autonomic management will be executed by the RSM and FM with fault properties. If a fault is classified that can be resolved at the runtime service level then it takes only service re-instantiation. In other case, we extend the fault handling mechanism to the application level, to FM, such that services can be reconfigured to utilize alternative services that provide the same or similar functionality as the service that caused the fault. Because of autonomic fault tolerance, a system maintains its level of reliability and availability, through reconfiguration in response to changes in its environment of execution.

3.2 Fault Management(FM)

When a fault cannot be resolved in the service manager level, the Wapee's fault manager reconfigures the application to utilize an alternative service that provides the same or similar functionality as the service that caused the fault. There are some

requirements of the application-level fault manager to ensure the functional reliability and continuity of an application:

- ***Functional consistency***: An alternative service must provide the same or similar functionality as the original one to achieve the consistent goal.
- ***Interoperability***: An alternative service must be interoperable with the adjacent services of the original one. Not only the interface-level interoperability, but also the semantic interoperability among the adjacent services must be ensured.
- ***Effectiveness***: An alternative service must be selected in a way that the service contributes to resolve the fault situation.
- ***Operational continuity***: The execution of an application must be continued after the reconfiguration of the application structure with an alternative service.

To meet these requirements, the fault manger in our framework supports description models to formally describe the types of fault conditions and the functionality of services. The fault manager also provides a service brokering mechanism that identifies a fault condition based on an exception event and service status, and finds alternative services that are interoperable with other services in an application and effectively resolve the fault condition.

Ontology-based Fault and Service Description Models. We have developed ontology-based description models to describe semantics of service faults and functionalities. We define three ontology hierarchies: the fault, service, and recovery strategy ontology. The fault ontology is for abstracting types of faults based on their causes such as the limitation of memory resource, and service errors. The fault ontology has a property to represent the resource condition that might cause a fault. The service ontology is for describing the functionality and resource requirements of a service. Finally, the recovery strategy ontology is for describing possible strategies to resolve a fault condition.

When an exception occurs in a service, the system reports the current status of the service and its environment. The service broker matches this fault information against the resource-condition property of the fault ontology to identify the corresponding fault semantics [6]. To find relevant fault semantics as much as possible, we adopt a semantic relaxation method, which, in an ontology hierarchy, collects nodes that have the same set of properties and are on a hierarchy of same subsumption – direct parents and children. Once a set of possible faults is identified, the service broker retrieves relevant recovery strategies to resolve the faults. The service broker then finds services that provide the same or similar functionality as the original service. A semantic relaxation method, which is similar to the method that we used for the fault ontology, is applied to the service ontology to extend the service set. The resource-requirement property of each service is then compared with the policy about resource property in each of the recovery strategies retrieved. Only the services that can contribute to resolve the fault (the services that meet the resource requirements) are selected as candidate services that can be used to substitute the original service.

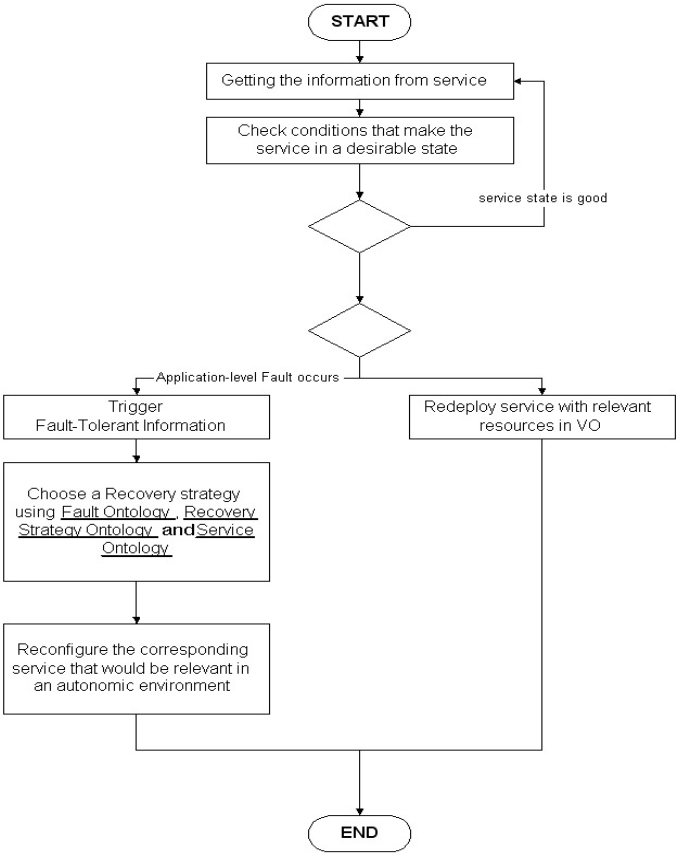


Fig. 3. The procedure of Service Management

4 Results

A prototype is to develop a workflow solution for complex grid applications to support the design, execution, monitoring, and performance visualization phases of development in a user-friendly way. We have developed a GUI based tool, Wapee Client, for workflow management, as shown in Fig. 4. A visual interface that allows for the graphical manipulation of workflow process instances provides a rich medium for the communication of dependencies and relationships between constituent jobs of a workflow process instance.

A job in workflow is represented by a set of interdependent tasks arranged in a Directed Acyclic Graph (DAG) [7]. After the creation of the DAG the resources identified in the workflow must be mapped onto the available grid resources [8]. The RSM supports run-time execution and job monitoring. Output results can also be available for a view from the Wapee Client.

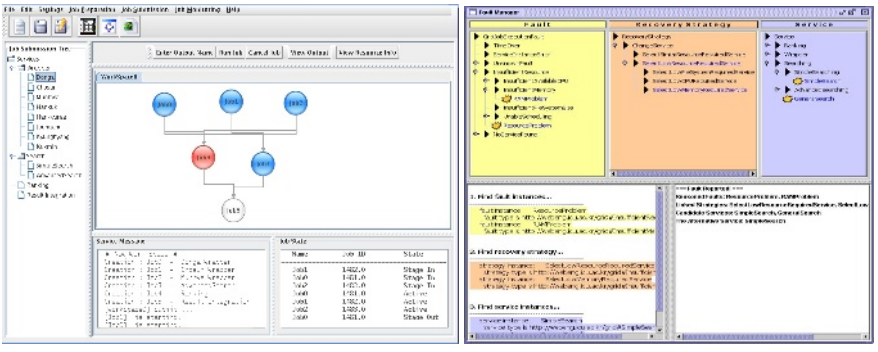


Fig. 4. Client Interface

Our main approach to autonomic service reconfiguration is performed in two steps. First, context-aware service discovery provides a set of services that are candidate to the configuration. Second, starting from the selected services and user task, context-aware process integration provides a set of configuration schemes that conform to the task's behavior further meeting all the context requirements.

We present a simple example that describes how our autonomic service reconfiguration algorithm can be used in a UbiComp environment. This example scenario is web-based applications, such as aggregation, searching and ranking about enormous web-based information. First, user can gather tremendous editorials on various newspaper website in the same breath using 'Wrapper Applications' of distinct type. Each 'Wrapper application' takes different time when it finishes. We choose three 'Wrapper Applications' for this experiment. And then, user can both view the result and send input-file for other applications at next phase. We select 'Ranking application' and 'Search application' for mid-applications of our experiment. The 'Search application' searches some words at forepart result. The 'Ranking application' finds selected word at forepart result and then shows ranking. Finally we join the whole information through different applications using 'Aggregation application'.

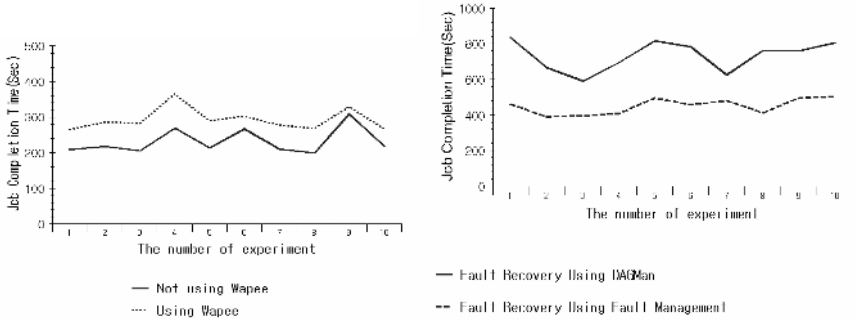
Fig. 5. Our test scenario

For example, when a fault occurs at 'Searching application' phase, Wapee analyze fault properties and classify the fault type, and replace another useful searching application using fault recovery strategy of FM. Our defined configuration property of searching application is shown in Fig. 5.

If fault occurs during using 'Advance Searching' application, we can overcome the fault using RSM and FM. If fault is classified that cannot be resolved at the runtime service manager level. To overcome such situation, we extend the fault handling mechanism to the application level, Fault Manager, such that the application can be reconfigured to utilize an alternative service that provides the same or similar functionality as the service that caused the fault. Its case is alterative service, 'Simple Searching'.

On Fig. 6 we showed the success rate and percentage of used fault-tolerance mechanism in Wapee. Wapee detect fault and recover them through Runtime Service Manager (RSM). The whole procedure takes about 326 seconds. This fault-tolerance mechanism is very basic algorithms that try to allocate resources on the nearest surrogate possible. If faults cannot be resolved at the service manager level then the RSM notify the fault handling information to the Fault Manager at application level. The whole procedure takes about 350 seconds, if Wapee detected these faults and recovered them using semantically Ontology, as shown in Fig. 6 (b).

These figures tell us that using fault recovery system, Wapee, increases service availability and executes resource efficiently in ubiquitous computing environments. It also shows us that the overhead ratio of middleware and application is kept in a relatively stable level (16.16% using RSM, 24.68% using FM) regardless of the variation of resource environment and service configurations. Our experiment validates the practicability and soundness of Wapee. The overhead of middleware is kept in a small ratio with respect to the overall system cost.



(a) (b)

Fig. 6. Performance Comparison using Wapee and non-Wapee of the Job

5 Conclusion and Future Works

Wapee with autonomic management executes likely faulty applications successfully with semantically meaningful strategies associating with service and fault ontology trees in ubiquitous environments. When a fault is found in runtime execution, Runtime Service Management (RSM) autonomically identifies the faults and decides if the fault might be resolved in runtime level or not. For resolvable faults in runtime, RSM configures Application Deployment Description again to obtain alternative

resources for the application. Otherwise, Application-level Fault Management supports dynamic software component reconfiguration plan based fault and service ontology to provide fault-tolerance in a ubiquitous computing environment.

Description Graph (ADG) with the help of the semantics of services and faults ontology; and informs the ADG for new deployment of the application autonomically. This allows better semantic interoperability between different context information on UbiComp environment. In addition, Wapee client, one of other strengths of Wapee, provides easy-of-use user interface for application construction, runtime execution, real-time monitoring and visualization of results.

For future work in Wapee, we are planning to implant an effective and autonomic meta-scheduler in collaboration with various local schedulers. Scheduling will be done with some consideration of application configuration information, environmental condition, user profile, and other special requirement such as fault tolerance policies to improve the quality of an application and resource utilization.

References

1. M. Weiser: The computer for the 21st Century Scientific American, Vol. 265, No. 3, pp. 94-104, September, 1991.
2. I. Foster. C. Kesselman, S. Tuecke: The Anatomy of the Grid: Enabling Scalable Virtual Organizations International J. Supercomputer Applications, 2001.
3. S. Tadepalli, C. Ribbens, S. Varadarahan: GEMS: A Job Management System for Fault Tolerant Grid Computing High Performance Computing Symposium, 2004
4. CORBA Fault: http://www.omg.org/cgi-bin/apps/doc?formal/01-09-29.pdf
5. Z. Kalbarczyk, R. K Iyer, L. Wang: Application Fault Tolerance with Armor Middleware Internet Computing, March/April 2005 (Vol 9, No 2) pp 28-37
6. Y. Hainning, E. Letha: Towards a semantic-based approach for software reusable component classification and Retrieval In Proceedings of the 42nd annual Southeast regional conference, 110-115, 2004
7. Condor DAGMan: http://www.cs.wisc.edu/condor/dagman/
8. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman: Grid Information Services for Distributed Resource Sharing Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001
9. P. Horn: Autonomic Computing: IBM's Perspective on the State of Information Technology, http://www.research.ibm.com/autonomic/manifesto/, 2001
10. J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke: Condor-G: A Computation Management Agent for Multi-Institutional Grids, Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10) San Francisco, California, August 7-9, 2001
11. B. Schilit, N. Adams, and R. Want: Context-aware computing applications. In IEEE Workshop on Mobile Computing Systems and Applications, 1994.

Remote Class Prefetching: Improving Performance of Java Applications on Grid Platforms

Yudith Cardinale, Jesús De Oliveira, and Carlos Figueira

Universidad Simón Bolívar,
Departamento de Computación y Tecnología de la Información,
Apartado 89000, Caracas 1080-A, Venezuela
{yudith, figueira}@ldc.usb.ve, jesus@bsc.co.ve

Abstract. In this paper we introduce and evaluate two prefetching techniques to improve the performance of Java applications executed on the grid. These techniques are experimentally evaluated on two grid environments, by running test applications on two different grid deployment configurations. Our testbed is SUMA/G, a grid platform specifically targeted at executing Java bytecode on Globus grids. The experimental results show that these techniques can be effective on improving the performance of applications run on the grid, especially for compute intensive scientific applications.

Keywords: Java applications, Class prefetching, Computational Grids, Distributed applications, Collaborative platforms.

1 Introduction

Distributed platforms for compute-intensive processing and resource sharing, increasingly known as grids [1], provide technologies for integrating multi-institutional sets of computational resources. Grid platforms increase the possibility of environments in which multiple users, geographically distant, may share data, pieces of software, computation resources, and even specialized devices [2].

The proliferation of grid platforms is fueling the development of distributed applications, which could be executed on remote sites. In these cases, application repositories (for instance, a client or user interface machine) and execution platforms typically reside on different sites, rising the need to transfer the application code from repositories to execution sites.

There has been increasing interest in supporting Java within the grid [3,4]. Java programs are organized as independent class files, each containing the methods and state variables that implement the class. Before any method can be executed or any state variable can be modified, the entire class must be transferred to the location where execution takes place.

Most Java Virtual Machines (JVM) implementations load classes on demand at the time of the first reference to each class. This is called *dynamic loading*.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 594–606, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Dynamic loading causes a delay on executions every time a class file load request is issued, since the thread triggering the load stalls until the class has been loaded, verified, resolved, and initialized. Hence, users experience these transfer delay intermittently during execution as well as upon program invocation.

One solution to the performance problem associated with dynamic loading is to load all the code at once. Hence, all transfer overhead appears before the application starts running, but afterward the application does not stall waiting for remote classes. However, it could result in loading (potentially many) classes that will not be needed, causing increased resources (e.g., bandwidth, CPU time, memory) overhead and therefore incurring in a performance penalty. It would be desirable to make sure that most of the transferred classes will be actually used, such as to avoid paying the costs for class files that won't be used. This is the execution model currently supported by most grid systems (e.g. those based on Globus, such as LCG [5] and TeraGrid [6]).

Another solution is to use *prefetching* for masking transfer delay by overlapping class loading with computation. Prefetching for Java has been used for improving performance in local execution contexts [7][8]. The goal is to have classes loaded from disk into memory (or from memory to cache) before they are referenced. The same principle has been used in the context of distributed computing. In [9], they propose two techniques, namely class prefetching and class splitting, in order to reduce remote class loading overhead involved in mobile programs execution. An alternative strategy, also used in the context of mobile programs, is offered in [10]. In this work, the authors propose to mask the transfer time, especially initial latency, by transferring blocks of code based on their semantic affinity. Thus, the program will be able to start executing while other blocks, needed later, are loaded.

Prefetching request must be made early enough so that the transfer delay is overlapped. The two goals of prefetching request are: i) to prefetch as early as possible to reduce (or eliminate) the delay when the actual reference is made; and ii) to ensure that prefetching is not made on a path which causes prefetching to be performed too early (or on a path that will not be executed) and may interfere with classes that are needed earlier than the class being prefetched. In this case, prefetching can introduce delays by using up available network bandwidth.

In this paper, we explore the use of prefetching techniques to improve the performance of Java applications on grids. Our testbed is SUMA/G [11], a grid platform specifically targeted at executing Java bytecode on Globus grids. In SUMA/G, all application's classes and files are loaded remotely on demand into execution platforms.

Prefetching techniques impact is assessed in two different grid environments supported in SUMA/G:

Single class repository: All application classes located on the user (client) machine. In this case, a modified version of the prefetching strategy proposed in [9] is used.

Multiple class repositories: Application classes located on the user machine and distributed repositories. Access to classes on distributed repositories is

provided by JaDiMa, a collaborative platform to build high performance Java applications on grid platforms [12]. The prefetching strategy used here is part of JaDiMa.

The main contribution of this paper are two prefetching techniques that improve the performance of Java applications on the grid. These techniques are experimentally evaluated on each grid environment, by running test applications on two different grid deployment configurations.

2 SUMA/G Overview

suma/g (Scientific Ubiquitous Metacomputing Architecture/Globus) [11] is a grid platform that transparently executes Java bytecode on remote machines. It extends the Java execution model to grid platforms; in particular, classes and data are dynamically loaded. suma/g middleware was originally built on top of commodity software and communication technologies, including Java and CORBA [13]. It has been gradually incorporating Globus general services by using the Java CoG Kit [3]. Hence, suma/g grids can be connected to deployed Globus based grids, as well as leverage on Globus technology evolution. suma/g architecture is depicted in Figure 1; its components are described below.

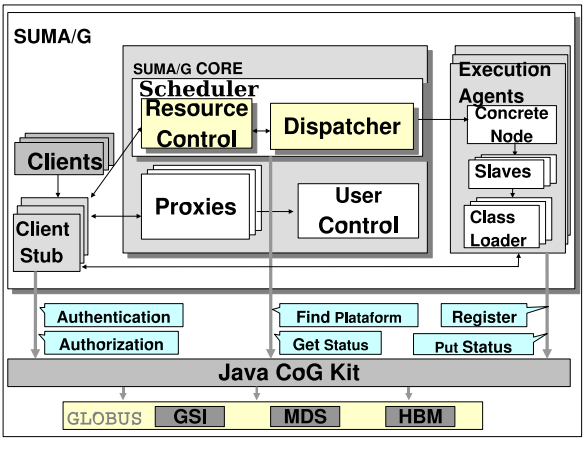


Fig. 1. suma/g Architecture

2.1 SUMA/G Components

In this section, we describe suma/g components, and their role in the execution of applications.

Proxy: Receives an object from `Client Stub`, containing application data such as the name of the main class, scheduling constraints (optional) and data structures to reduce the number of communications (optional); these are called *preloaded data*. After checking user permissions, the `Proxy` asks the `Scheduler` for

a suitable execution platform, then sends the application object to the selected one. In case of submitting off-line jobs, the `Proxy` keeps results until the user requests them.

Scheduler: Responds to `Proxy` requests based on the application requirements and status information obtained from the grid platform. Using the Globus MDS service, the `Scheduler` learns of grid resources, obtaining information about available execution platforms (including memory size, available libraries and average load), data sets hosted at specific locations, and so on. With this information, the `Scheduler` selects a suitable resource satisfying the application requirements, while looking for load balance in the grid.

User Control: It is in charge of user registration and authentication. The GSI is used for user authentication and authorization in SUMA/G, as well as a mechanism for including all SUMA/G components in the grid security space.

Client Stub: It creates the application object, retrieves results and performance data, and serves `Execution Agent` requests (callbacks) to load classes and data dynamically. It is executed on the user machine or on a SUMA/G entry server. In any case, the user must have a valid certificate installed on that machine.

Execution Agent: On starting, it registers itself at the `Scheduler` as a new available resource. During operation, it receives the application object from the `Proxy` and launches execution, loading classes and files dynamically from the client or from a remote file system through the SUMA/G class loader and the SUMA/G I/O subsystem. Once the application has finished, the `Execution Agent` sends the results back to the client. In a parallel platform, it plays the role of the front-end. Only the front-end of a parallel platform is registered on SUMA/G either as an mpiJava enabled platform or as a farm, for multiple independent job executions.

2.2 SUMA/G Execution Model

The basics of executing Java programs in SUMA/G are simple. The users can start the execution of programs through a shell running on the client machine. They can invoke either `Execute`, corresponding to the on-line execution mode, or `Submit`, which allows for off-line execution (batch jobs). At this time a proxy credential is generated (by using GSI) that allows processes created on behalf of the user to acquire resources, without additional user intervention. Once the SUMA/G `CORE` receives the request from the client machine, it authenticates the user (through GSI), transparently finds a platform for execution (by querying the MDS), and sends a request message to that platform. An `Execution Agent` at the designated platform receives an object representing the application and starts, in an independent JVM, an `Execution Agent Slave`, who actually executes the application. The SUMA/G Class Loader is started in that new JVM, whose function is to load classes and data during execution. Supported classes and input files sources, and output destinations, include: a) the machine (client) where the application execution command is run and, b) a remote file server on which the user has an account. A pluggable schema allows for implementing

several protocols to manage remote files access. Currently, schemas for CORBA and sftp are available; support for gridFTP and others will also be provided.

To execute an application, either on-line or off-line, the user has only to specify the main class name. In the case of `Execute` service, the rest of the classes and data files are loaded at run-time, on demand, without user intervention. Standard input and output are handled transparently, as if the user were running the application on the local machine. For the `Submit` service, SUMA/G `Client` transparently packs all classes together with input files and delivers them to SUMA/G `CORE`; the output is kept in SUMA/G until the user requests it.

3 Prefetching on *Single Class Repository* Environments

SUMA/G prefetching is inspired on an algorithm proposed in [9]. It is based on the definition of a graph representing the control flow of classes instantiation. In what follows, it will be called *Classes Instantiation Graph* (CIGRAPH). This graph is built according to the *first use* reference to classes. Following the usual rules for dynamic loading, a *first use* reference will cause a non-local class file to be transferred.

In the original algorithm, prefetching decisions are made before execution by analyzing the CIGRAPH and instrumenting the original code with prefetching requests for classes being referenced. The prefetching decisions are based on the CIGRAPH, the estimated *first execution* and transfer times of each class. The estimated times are computed from previous executions profiles. The *first execution* time determines the order in which *first use* references are processed and it is measured from the start of the application.

We made a simpler implementation of that algorithm. Our implementation is based on the CIGRAPH, but we do not instrument the original code. Instead, we implemented a module which analyzes the graph, and makes decisions about prefetching at execution time. The module was implemented as a thread in order to overlap computation with class transfers. When an application is executed for the first time in SUMA/G, the bytecode is parsed in order to extract the information needed to create the CIGRAPH. This original CIGRAPH does not contain any information about transfer and *first execution* times. This information will be obtained on further executions. We use Espresso [14] to parse the code.

With the original CIGRAPH, all classes are prefetched according to a depth first search order on the CIGRAPH. If the CIGRAPH has information about transfer and *first execution* times, prefetching takes them into account. For each class A in CIGRAPH, but the main class, its $Time\_Left$ is computed as

$$Time\_Left(A) = first\_execution\_time(A) - transfer\_time(A) \qquad (1)$$

Class A is prefetched only if $Time\_Left > 0$. Figure 2 shows an example of a CIGRAPH used for prefetching request decisions: classes B, C and D are chosen for prefetching because their $Time\_Left$ is greater than 0.

Figure 3 shows the steps for executing an application with prefetching support on SUMA/G. On the client side, when a user requests an application execution,

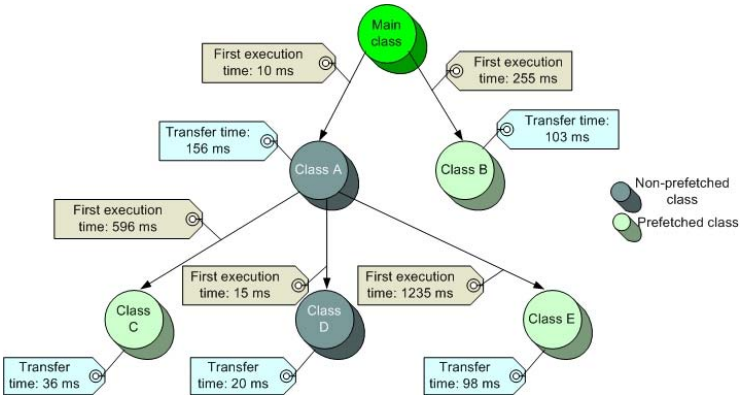


Fig. 2. Example of CIGRAPH used for prefetching request decisions

the `Client Stub` parses the application bytecode building a CIGRAPH for the entire application. This is made only the first time the application runs on SUMA/G. On further executions, `Client Stub` directly uses an already available CIGRAPH, obtained from previous executions (step 1).

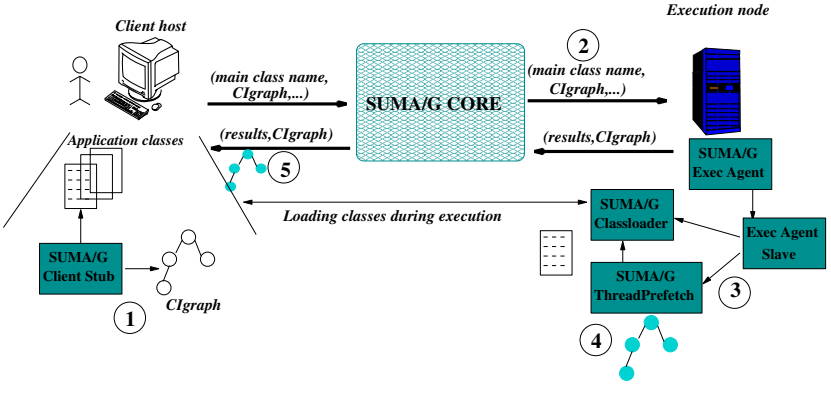


Fig. 3. Application execution on SUMA/G with prefetching support

After the SUMA/G client invokes the remote execution, it obtains a node with prefetching support, and sends the CIGRAPH along with the main class reference to the `Execution Agent` at the selected node (step 2). Once this `Execution Agent` receives the execution request, it starts an `Execution Agent Slave` in a separated JVM (step 3) in order to execute the application. Two service threads are started initially within this JVM:

1. `SUMAgClassLoader`, whose purpose is to load classes and data on demand from the client
2. `SUMAgThreadPrefetch`, which will prefetch classes during execution.

The main class is then directly loaded from the client and started. Concurrently, `SUMAgThreadPrefetch` decides which classes to prefetch according to the information in the CIGRAPH. The graph is updated with *first execution* and transfer times, measured during current execution (step 4). To prefetch classes, `SUMAgThreadPrefetch` uses the `SUMAgClassLoader` `loadClass` method.

When execution finishes, the `Execution Agent` passes this information back to the SUMA/G core, which returns the results to the client, including the updated CIGRAPH (step 5).

4 Prefetching on *Multiple Class Repositories* Environments

Multiple Class Repositories is supported in SUMA/G by JADIMA. JADIMA [15] [1] is a collaborative platform that aims to support high performance distributed Java applications development and execution. Its main goal is to allow applications to transparently instantiate classes located at remote repositories. JADIMA allows for compilation and execution of high performance JAVA applications that use distributed software components without keeping them locally in the application developer machines. In this sense, JADIMA lets developers use a much smaller representation of the libraries (referred to as *stubs*) which the application depends on during the compilation process. *Stubs* are generated automatically by JADIMA and substitute actual libraries. When a remote class is instantiated for the first time during execution, the JADIMA class loader locates a repository hosting the actual class bytecode; the class is then transferred from the repository to the execution machine, and execution is resumed. The use of a version numbering convention allows for automatic updating of new versions of used libraries without affecting its correct execution.

JADIMA implements a simple prefetching strategy in order to reduce the impact on execution time of remote class loading. Recall that pure Java model consists of loading classes on-demand, one by one. JADIMA goal is to try to have some of the remote classes loaded before they are instantiated, such that application's idle time waiting for class transfers is diminished.

The prefetching strategy consists on grouping remote classes according to its invocation time: whenever a remote class of a group (called cluster) is instantiated for the first time, all members of the group are requested and loaded from their respective remote repository, including the instantiated class itself. Clusters are computed and adjusted at execution time. The clustering algorithm is described below.

4.1 Class Clustering

Classes are grouped according to *temporal proximity*. Temporal proximity computation is based on the *effective time* (defined below) where each remote class is

[1] http://sourceforge.net/projects/jadima/

instantiated for the first time (*first execution* time). *First execution* time for all remote classes instantiated are registered and saved by the JaDiMa class loader during execution; this information will be used on next application's executions.

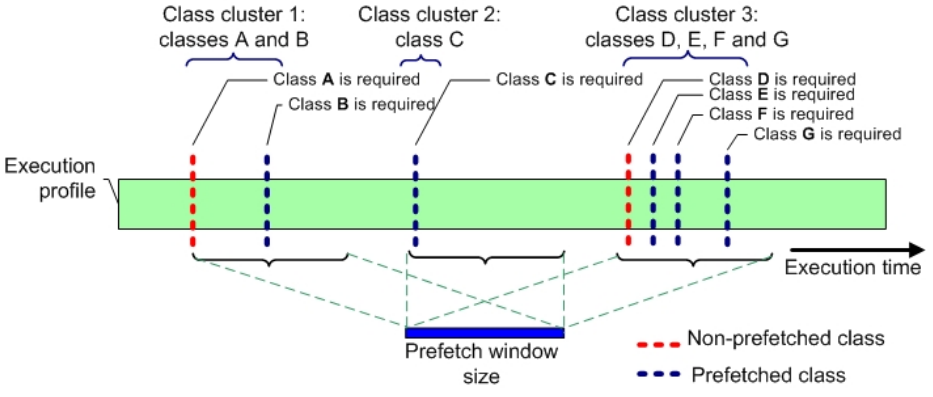


Fig. 4. JaDiMa class clustering

The effective time (*ETime*) of a class is the time, measured from the application's start time, of its *first use* reference, ignoring the time spent transferring previously instantiated remote classes. Starting with the first remote class instantiated (i.e., the `main` class) at execution time, the algorithm creates the clusters. Let's call C_{Gx} the first class of a cluster G_x, and $ETime(C_{Gx})$ its effective time. Every succesively instantiated remote class C_i not yet added to a cluster and such that

$$(ETime(C_i) - ETime(C_{Gx})) < \varepsilon$$

is added to cluster G_x. Clustering depends on ε, the *Prefetch window size* parameter. The first remote class (not yet in a cluster) whose *ETime* is more than ε from $ETime(C_{Gx})$ become the first class of a new cluster; the procedure iterates until all classes are grouped into clusters. Figure 4 depicts class clustering.

4.2 Implementation of JaDiMa Prefetching in SUMA/G

JaDiMa has been integrated to suma/g [11]. The suma/g `Execution Agent` uses JaDiMa class loader, which takes care of remote classes instantiations; it includes the mechanisms to compute an application's classes clusters during execution, according to the rules explained above (section 4.1). It uses two different threads to handle remote class transfers: one for the requested class, and another for the rest of the members of the cluster.

During the first execution of an application, the time at which every class is referenced, measured from the beginning of execution, is recorded. When the

application finishes, this information, for all referenced classes, is sent to the `Client`. The next time the application is executed, the previously recorded information is sent along with the execution request.

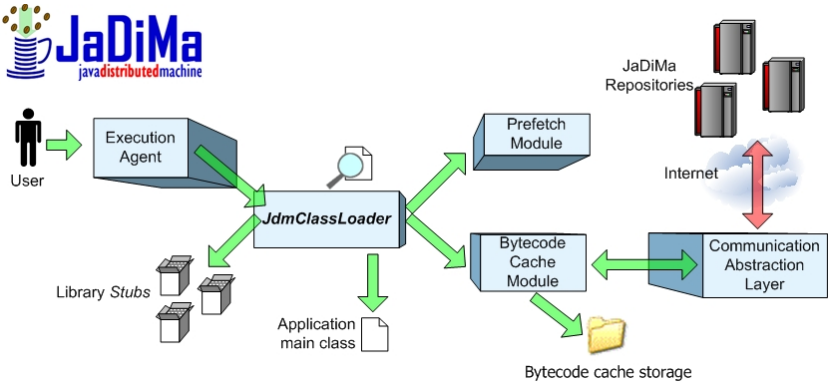


Fig. 5. JaDiMa execution model on SUMA/G

The JaDiMa prefetching architecture is depicted in figure 5. Whenever a remote class is instantiated:

1. JaDiMa class loader asks the *Prefetch Module* to compute the requested class' cluster using the *Prefetch window size* specified for this execution
2. the list of cluster members is then passed to the *Bytecode cache module*, which checks for this cluster's classes local availability (in memory or cache)
3. those classes that are not locally available are requested remotely by JaDiMa *Communication Abstraction Layer* (CAL) module.

The CAL module handles remote classes requests using two concurrent threads: one for the class instantiated by the application, and another for its cluster companions. Thus, whenever the instantiated class is loaded, the application can resume execution, while other prefetched classes are transferred.

The prefetching of a class is triggered by the first instantiation of any member of the cluster it belongs to. If the application is sequential (only one thread), the execution will stall until the instantiated class is transferred. Other members of the same cluster will proceed immediately, if the prefetching is complete before their *first use*. If the application is multi-threaded, a better overlapping of execution and class transfer might be expected.

5 Experimental Results

The prefetching techniques are experimentally evaluated for each grid environment by running test applications under two different grid deployment configurations:

- **LAN**: All SUMA/G components run on machines of a laboratory in our campus.
- **WAN**: SUMA/G `CORE` and `Execution Agent` run on machines at one site (same laboratory as above), and the `Client` run on a machine located outside the campus (25 Kms from the lab.)

The Java applications used in our experiments are JUNG [16] (*Java Universal Network/Graph*), and a subset of the JavaGrande benchmark suite [17]. Both applications are sequential, single threaded. It means that, when a class is referenced and is not locally available, the execution stalls until the class is transferred.

JUNG is an open-source software library that provides a language for modeling, analysis, and visualization of graphs and network data. JUNG is composed by several packages, the main ones being `COLT` (http://cern.ch/hoschek/colt/), `commons-collections` (http://jakarta.apache.org/commons/collections/), and `xerces` (http://xml.apache.org/). The number of classes in all packages amounts to over 200, but only 119 are actually referenced in our tests.

The JavaGrande Benchmark is a suite of sequential and parallel Java applications having large requirements for any or all of: memory, bandwidth and processing power. It includes computational science and engineering codes, as well as large scale database applications and business and financial models. For this experiments, we chose the sequential subset of application in section 2, with size A (small data size). All seven applications in section 2 are executed sequentially, from a main class, which also uses some utility classes. In total, the application uses 18 classes.

The platforms used for the experiments are Linux based PCs:

- SUMA/G `CORE` and an `Execution Agent` run each on a Pentium 4 3.4 GHz, 1GB memory, with Debian Sarge (kernel 2.4.27). Both machines are on the same 100 Mbps LAN.
- In the WAN configuration, the SUMA/G `Client` runs on a Pentium 4 2.2GHz, 1GB memory, connected to the Internet by a 768 Kbps DSL link. The connection of the lab machines to the Internet shares an 8 Mbps link.
- In the LAN configuration, the SUMA/G `Client` runs on an AMD Athlon XP 1800, 768 MB memory, which is on the same LAN as the `CORE` and `Client`.

The results for each grid environment are shown below. The test applications are run in LAN and in WAN configurations, both with and without prefetching. The results shown are average execution times (out of 10 runs) and the improvement (speed up) obtained using prefetching with respect to not using prefetching.

5.1 Single Class Repository Results

In this environment, all classes are loaded from the client machine into the execution machine. Tables 1 and 2 show the results for, respectively, JavaGrande and JUNG applications.

In the case of JavaGrande, we get performance improvement both in LAN and WAN grid configurations. Prefetching is effective since every class has an execution time comparable to transfer times, such that some execution time and transfer time overlapping is possible. Note the improvement is larger for WAN than for LAN. When prefetching is not used, every class reference is transferred from the SUMA/G `Execution Agent` to the `Client`. Since connection times take longer in WAN than in LAN, a better overlapping between execution and class transfers is obtained.

Table 1. Average execution times and speed up for JavaGrande with Single Class Repository

	No prefetching	**Prefetching**	*Speed Up*
LAN	36 sec.	35 sec.	3%
WAN	104 sec.	81 sec.	22%

Table 2. Average execution times and speed up for JUNG with Single Class Repository

	No prefetching	**Prefetching**	*Speed Up*
LAN	69 sec.	71 sec.	-2%
WAN	172 sec.	187 sec.	-9%

For JUNG, results with prefetching are worse both in LAN and WAN. JUNG uses many classes, but each one does little computation. Hence, the analysis of the CIGRAPH results in that almost none of the classes is prefetched. Performance degradation is mostly due to the resources used up by the prefetching thread.

5.2 Multiple Class Repositories Results

In this environment, most of the classes are loaded from two classes repositories, and the rest from the client machine. The repositories are located close to the execution platform, actually in the same network. Tables 3 and 4 show the results for, respectively, JavaGrande and JUNG applications.

In the case of JavaGrande, a performance improvement is again obtained, due to the combination of few classes to transfer and each one doing considerable computing work.

For JUNG, we obtain a considerable performance improvement in the WAN environment, due mostly to overlapping computing with classes transferred from the client. In LAN environment, since transmission times from the client are shorter, no improvement is gained.

5.3 Results Discussion

For the JavaGrande application, performance improvement was obtained with prefetching in all cases, even though we ran the benchmark using the small size

Table 3. Average execution times and speed up for JavaGrande with Multiple Class Repositories

	No prefetching	**Prefetching**	*Speed Up*
LAN	39 sec.	37 sec.	5%
WAN	54 sec.	50 sec.	7%

Table 4. Average execution times and speed up for JUNG with Multiple Class Repositories

	No prefetching	**Prefetching**	*Speed Up*
LAN	73 sec.	73 sec.	0%
WAN	103 sec.	89 sec.	13%

data set. It is expected to have even better performance improvement with more computing intensive applications.

JUNG represents a more challenging problem for prefetching strategies. In our experiments, performance was degraded in the Single Class Repository environment. An improvement was achieved in the Multiple Class Repositories environment due to the proximity of the repository to the execution agent. While this is a desirable characteristic to take into account to locate class repositories (to be close to the execution platforms), clearly it won't always be possible to do so.

6 Conclusions and Future Work

Java is a well established language in the programmers community, and it offers a number of advantages for scientific applications on distributed environments. As grids are becoming the platform of choice to execute distributed scientific applications, a suitable support for running Java applications on grids must be provided.

Prefetching is an interesting technique for overcoming performance degradations due to Java dynamic class loading in a distributed environment. It allows for masking the remote classes transfer time by loading classes before they are actually referenced.

In this paper we presented two prefetching techniques implemented on a grid platform called SUMA/G, which supports Java execution model. The experimental results show that these techniques can be effective on improving the performance of applications run on the grid, especially for compute intensive scientific applications.

Plans for future work include conducting experiments with other kinds of applications (e.g., different combinations of number of classes and computational complexity of classes), as well as trying other grid deployment scenarios.

References

1. Foster, I., Kesselman, C.: Computational Grids. In: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, Inc. (1999) 15–51
2. Berman, F., Fox, G., Hey, A., eds.: Grid Computing: Making the Global Infrastructure a Reality. Wiley (2003)
3. von Laszewski, G., Foster, I., Gawor, J., Smith, W., Tuecke, S.: CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids. In: ACM Java Grande 2000 Conference, San Francisco, CA (2000) 97–106
4. S. Tuecke and K. Czajkowski and I. Foster and J. Frey and S. Graham and C. Kesselman and T. Maguire and T. Sandholm and P. Vanderbilt and D. Snelling: Open Grid Services Infrastructure (OGSI) Version 1.0 (2003) Global Grid Forum Draft Recommendation.
5. LCG Team: LCG: Worldwide LHC Computing Grid. http://lcg.web.cern.ch/lcg/ (2006)
6. TeraGrid: http://www.teragrid.org (2006)
7. Cahoon, B., McKinley, K.: Tolerating latency by prefetching java objects. In: Proceedings of Workshop on Hardware Support for Objects and Microarchitectures for Java. (1999)
8. Krintz, C.J., Grove, D., Sarkar, V., Calder, B.: Reducing the overhead of dynamic compilation. Software Practice and Experience **31**(8) (2001) 717–738
9. Krintz, C., Calder, B., Hölzle, U.: Reducing transfer delay using java class file splitting and prefetching. In: Proceedings of the 14th annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications. (1999)
10. Stoops, L., Mens, T., D'Hondt, T.: Fine-grained interlaced code loading for mobile systems. In: Proceedings of ECOOPWS. (2002)
11. Cardinale, Y., Hernández, E.: Parallel Checkpointing on a Grid-enabled Java Platform. Lecture Notes in Computer Science **3470**(EGC2005) (2005) 741 – 750
12. Cardinale, Y., Blanco, E., Oliveira, J.D.: JaDiMa: Java Applications Distributed Management on Grid Platforms. Lecture Notes in Computer Science (2006) To appear.
13. Cardinale, Y., Curiel, M., Figueira, C., García, P., Hernández, E.: Implementation of a CORBA-based metacomputing system. Lecture Notes in Computer Science **2110** (2001) Workshop on Java in High Performance Computing.
14. Doerig, K.: Espresso, a Java compiler written in Java (2006) http://types.bu.edu/Espresso/report/Espresso.html.
15. Cardinale, Y., Blanco, E., DeOliveira, J.: JaDiMa: Arquitectura de Máquina Virtual para la Construcción de Aplicaciones JAVA en Plataformas Grids. In: XXXI Conferencia Latinoamericana de Informática (CLEI-2005), Colombia (2005)
16. O'Madadhain, J., Fisher, D., Nelson, T., Krefeldt, J.: JUNG: Java Universal Network/Graph Framework (2003) http://jung.sourceforge.net/index.html.
17. EPCC: The Java Grande Forum Benchmark Suite. http://www.epcc.ed.ac.uk/-javagrande (2006)

Using Data Consistency Protocol in Distributed Computing Environments*

Jaechun No[1], Chang Won Park[2], and Sung Soon Park[3]

[1] Dept. of Computer Software
College of Electronics and Information Engineering
Sejong University, Seoul, Korea
[2] Intelligent IT System Research Center
Korea Electronics Technology Institute
Bundang-gu, Seongnam-si, Korea
[3] Dept. of Computer Science & Engineering
College of Science and Engineering
Anyang University, Anyang, Korea

Abstract. Large-scale, data-intensive computing requires a sophisticated technology to be integrated with distributed file systems to provide clients with reliable and scalable high-performance accesses to the stored data. The clients physically share storage devices connected via a network like GigaEthernet or Fibre Channel and, on those clients, distributed file systems take responsibility for providing coordinated accesses and consistent views of shared data. In such a distributed computing environment, one of the major issues in achieving substantial I/O performance and scalability is to build an efficient locking protocol. In this paper, we present a distributed locking protocol that enables multiple nodes to simultaneously write their data to distinct data portions of a file, while providing the consistent view of client cached data. We conclude with an evaluation of the performance of our locking protocol.

1 Introduction

Large-scale, data-intensive computing requires a sophisticated technology to be integrated with distributed file systems to provide clients with efficient and scalable high-performance accesses to stored data. The clients are physically connected to one or more servers via a network like GigaEthernet or Fibre Channel [1,2,4,6] and, on those clients, distributed file systems take responsibility for providing coordinated accesses to remotely stored data and for providing consistent views of client cached data. In such a distributed computing environment, one of major considerations in achieving substantial I/O performance and scalability is to build an efficient locking protocol.

One of the general locking protocols for a distributed environment is to provide a token-based lock manager, as described in [1,4]. The basic idea behind the token-based lock manager is that before a client performs file data or metadata

* This work was supported in part by a Seoul R&BD program.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 607–617, 2006.
© Springer-Verlag Berlin Heidelberg 2006

operations, it requires a related lock from the lock server. If there is no conflicting request to the lock, or the client is the only requester to the lock, the lock server then grants the lock to the client.

A locking protocol to support data consistency and cache coherency has a significant effect on generating high performance I/O. For example, large-scale scientific applications in physics, chemistry, biology, and other sciences generate huge amounts of data and utilize them for data analysis, visualization, and so on. In order to achieve high-performance I/O, many such applications use parallel I/O methods where multiple client nodes simultaneously perform their I/O operations. MPI-IO is among those parallel I/O methods.

MPI-IO [5] is specifically designed to enable the optimizations that are critical for high-performance parallel I/O. Examples of these optimizations include collective I/O, the ability to access noncontiguous data sets, and the ability to pass hints to the implementation about access patterns, file-striping parameters, and so on. In order to achieve high I/O performance using MPI-IO on top of distributed file systems, the file system must provide the ability to lock a file per data section to have multiple concurrent writers to a file.

However, many of the locking protocols integrated with distributed file systems are based on a coarse-grained method [1,2] where only a single client at any given time is allowed to write its data to a file, while the other clients are waiting for the current node to finish its write operation even when the others would write to the different data portions of the same file. This drawback significantly degrades I/O performance in many scientific applications where supporting parallel write operations happens to be proved generating high I/O bandwidth.

In this paper, we present a distributed locking protocol based on multiple reader/single writer semantics for a data portion to be accessed. In this scheme, a single lock is used to synchronize concurrent accesses to a data portion of a file. However, several nodes can simultaneously run on the district data sections in order to support data concurrency. We conclude our paper by discussing performance evaluation of our locking protocol.

2 Design Motivation

Our main objectives in developing a distributed locking protocol were to provide high-performance parallel I/O, to minimize the communication latency occurred during the lock negotiation steps, and to utilize local lock services as much as possible.

- **High-performance I/O.** We designed the distributed locking protocol capable of allowing multiple concurrent writers to the same file to achieve high performance I/O. Also, the locking protocol provides data consistency between the data stored in the storage device and the data stored in the client-side cache. On top of the distributed file system integrated with this locking protocol, many data-intensive applications can generate high I/O bandwidth using parallel I/O libraries, such as MPI-IO.

– **Low communication latency.** We designed the locking protocol to reduce the network overhead taking place during the lock negotiation steps with Global Lock Manager (GLM). All the lock requests coming from the client nodes are evenly distributed on multiple GLMs. Moreover, in order to minimize the number of callback messages necessary to revoke and release a lock, we grouped all the client nodes into several node groups. If GLM finds the node group where the lock holder belongs to it then sends a lock revocation message to the node group. After the lock holder completes the corresponding callback function to release the requested lock, it sends back an acknowledgement to GLM to grant the lock to the requesting client node.
– **Use of local lock service.** we designed the locking protocol to utilize local lock services as much as possible in order not to cause communication overheads with GLM. In order to use the local lock service to the maximum extents, we designed the distributed lock scheme using the lazy-revocation or sticky lock method [1] to retain privileges on data sections, even in the absence of active processes on a client node. If process on a node accesses to the smaller data section than the section controlled by an already acquired lock and if the requesting lock mode does not conflict with the mode of the acquired lock, the acquired lock is then split and the lock of the requesting data section, called childlock, is returned to the process. However, the lock information of a childlock needs not be stored in GLM.

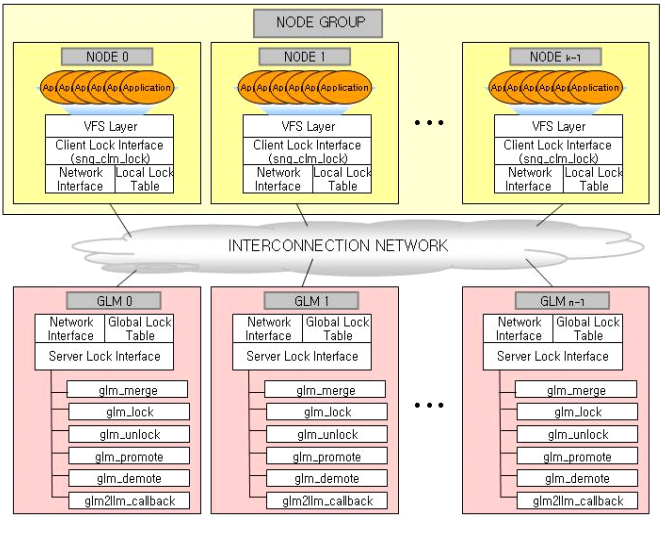


Fig. 1. A distributed lock interface

3 Implementation Details

3.1 Software Architecture of Distributed Locking Protocol

Figure 1 illustrates the distributed lock interface that is integrated with distributed file systems. Applications issue I/O requests using the local file system interface, on top of VFS layer. Before performing an I/O request, each client should acquire an appropriate distributed lock from GLM in order to maintain data consistency between the cached data on clients and the remote, shared data on servers. The lock request is initiated by calling the lock interface, `snq_clm_lock`. The information of the lock acquirement for clients is stored in the local lock table that is created per node.

As mentioned in section 2, in order to reduce the communication latency occurring at the lock acquire step, we grouped the client nodes into several node groups. In the current implementation, an eight bit integer is used to denote node groups. When a client acquires an appropriate lock to perform I/O operation, the bit corresponding to the node group where the client belongs to is set to 1.

When a client requests a lock to GLM, GLM first locates the node group where the lock holder belongs to and then sends a callback message to the nodes of the node group. When the lock holder receives the callback message, it releases the requested lock and sends back an acknowledgement to GLM to grant the lock to the requester.

After a client acquires the desired lock for I/O, it sends the file metadata, such as file size, modification time, access time, and data block addresses, to the first GLM, `GLM0`, to modify the corresponding file inode. Figure 1 shows the necessary GLM functions to serve the client lock requests.

3.2 Hierarchical Lock Layer

Figure 2 represents a hierarchical overview of the locking construct with two client nodes and one GLM. The lock modes that we provide for are SHARED for multiple read processes and EXCLUSIVE for a single write process. The lock structure consists of three levels: metalock, datalock, and childlock.

The metalocks, `inode0` on node A and `inode1` on node B, synchronize accesses to files and the value of a metalock is an inode number of the corresponding file. Below the metalock is a datalock responsible for coordinating accesses to a data portion. For example, on node A, metalock `inode0` is split into two datalocks associated with the data sections `0000-3FFF` and `4000-5FFF` in bytes and, on node B, two datalocks below `inode1` are associated with the data sections `0000-2FFF` and `3000-4FFF` in bytes. In order to grant a datalock, the lock mode of the higher lock (metalock) must be SHARED, meaning that a file is shared between multiple clients.

The childlock which is a split datalock sits at the lowest level. As mentioned in section 2, given that a datalock is granted, the datalock can be split further to maximize local lock services as long as the data section to be accessed by a requesting process does not exceed the data section of the datalock. In Figure 2, the datalock for the data portion `0000-3FFF` on node A is split into three

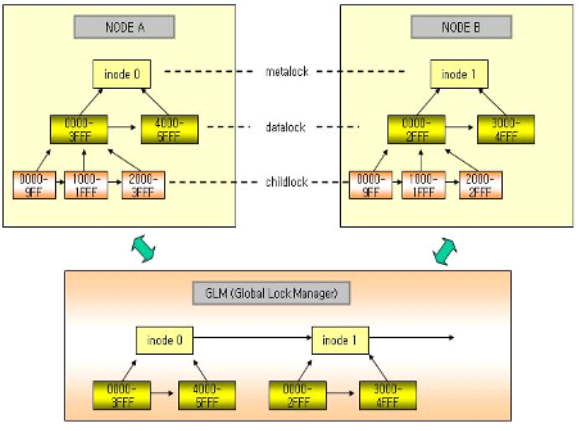


Fig. 2. A hierarchical overview of distributed locking protocol

childlocks that control accesses to the data portions `0000-9FF`, `1000-1FFF`, and `2000-3FFF`, respectively.

The childlock is locally granted and therefore the requesting process needs not communicate with GLM to obtain the childlock. However, the childlock is granted only when the lock mode of a childlock is compatible with that of the higher datalock. The datalock and the childlock are located by comparing the starting file offset and data length being passed from the local file interface.

GLM contains the global lock information consisting of a list of the locks that each GLM is responsible for serving. In Figure 2, GLM contains the metalocks, `inode0` and `inode1`, and the datalocks of the data portions `0000-3FFF` and `4000-5FFF` of `inode0` and of the data portions `0000-2FFF` and `3000-4FFF` of `inode1`. GLM also contains the node group information indicating those groups where the lock holders belong to.

3.3 Lock Management

Figure 3 describes the steps taken by three clients to acquire their desired locks. Suppose that node A retains a metalock, `inode0`, with EXCLUSIVE mode. At time t_i, a client on node B requests a write lock to GLM for the data range `0000-3FFF` of `inode0` and the other client on node C requests a write lock for the data range `4000-7FFF` of the same file. GLM sends to the lock holder on node A a lock revocation message which changes the lock mode to NOLOCK after the lock release.

GLM grants the write lock for the data range `0000-3FFF` to the client on node B and the write lock for the data range `4000-7FFF` to the client on node C. In order to allow parallel write operations on the same file, the metalock, `inode0`, held by both clients is changed to the lock mode SHARED.

The datalocks on the clients can be split to the childlocks as long as the I/O requests associated with the childlocks do not conflict with the parent's lock

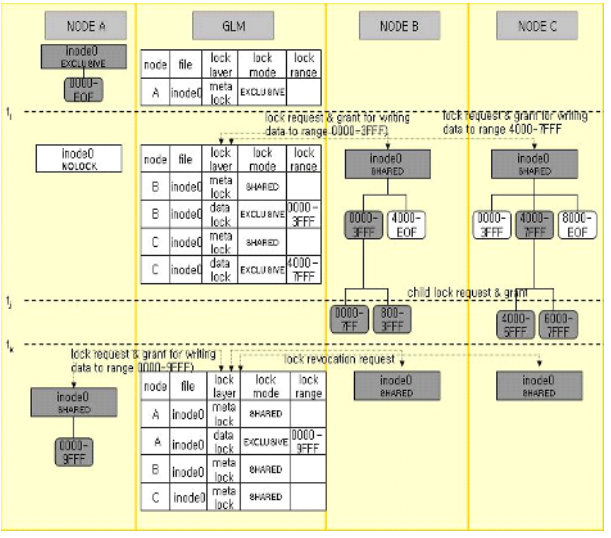


Fig. 3. Distributed lock management

mode. For example, at time t_j, because two write operations are locally invoked on each node and the requested lock mode (EXCLUSIVE) does not conflict with the parent lock mode (EXCLUSIVE), the datalock on node B is split to two childlocks, `0000-7FF` and `800-3FFF`, and the datalock on node C is split to another two childlocks, `4000-5FFF` and `6000-7FFF`. Note that the information about those childlock acquirements is not stored in GLM.

At time t_k, when a client on node A asks GLM for a write lock to write data to the data range `0000-9FFF`, the data locks and the childlocks on node B and on node C are released, while keeping the metalocks on both nodes with SHARED mode.

In order to avoid parallel writes and reads on the same file to be serialized to access the same file metadata, we merge all the metadata being accessed on `GLM0` which is attached to the storage disk. When a client acquires a metalock to access to a file at the first time, it would also receive the associated file metadata from `GLM0`, along with the metalock. After finishing the I/O operation related to the acquired lock, the client sends the file metadata to `GLM0` to merge them to maintain the consistent view of the file to all nodes. By doing this way, each client can independently perform its I/O operation as possible, without considering other clients accessing the same file.

3.4 Function Calls

Figure 4 represents the functions to be called to serve the lock request, lock release, and lock grant operations. The lock request operation is started by calling `snq_clm_lock` in the local file interface to read or write data. Once process finishes its I/O operation, `snq_clm_unlock` is called to wake up a sleeping process, if any, blocked while waiting for the lock to be released.

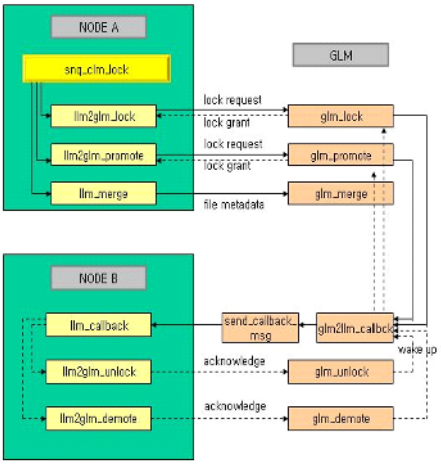


Fig. 4. Steps to acquire a distributed lock

If the requesting lock exists in the local lock table or the data portion to be accessed does not exceed the data portion of an already acquired lock then the request is immediately satisfied and the lock is returned to process. If the requesting lock does not exist in the local lock table, indicating that either it is already held by a different node or the lock is requested at the first time, then `llm2glm_lock` is called to communicate with GLM. If the requesting lock mode is EXCLUSIVE, then `llm2glm_promote` is called to upgrade the lock mode.

GLM receives the lock service request and then calls `glm_lock` or `glm_promote` to grant a lock or to upgrade lock mode. `Glm_lock` and `glm_promote` both call a callback invoke function, `glm2llm_callback`, to send an appropriate callback message to remote clients. `Glm2llm_callback` invokes `send_callback_msg` that sends a message to the node group where the lock holder belongs to. After invoking `send_callback_msg`, `glm2llm_callback` is blocked until it is woken up by `glm_unlock` or by `glm_demote`.

`Glm_unlock` is a function to be called to update the global information of the lock that has been released on a remote lock holder and `glm_demote` is a function to downgrade a lock of a remote lock holder.

On a client node, once a callback message is received, the lock interface calls `llm_callback` to release or to downgrade the lock requested. The lock release operation is performed by calling `llm2glm_unlock` and the lock downgrade operation is performed by calling `llm2glm_demote`. After completing its intended operation, each function sends back an acknowledgement to GLM to grant the lock to the requesting node.

When an I/O operation is completed, `snq_clm_unlock` invokes `llm_merge` to send all the associated file metadata to `GLM0`. `Glm_merge` collects the file metadata from clients and writes them to the attached storage.

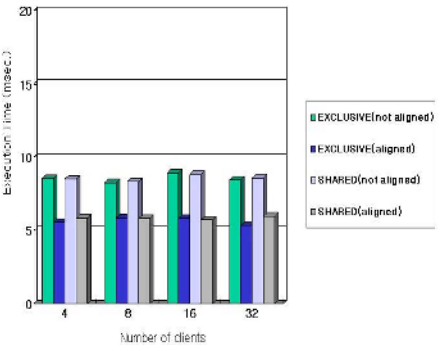


Fig. 5. Time overhead to acquire a distributed lock using one GLM. Each client read from or wrote 0.8Mbytes of data to the distinct section of the same file.

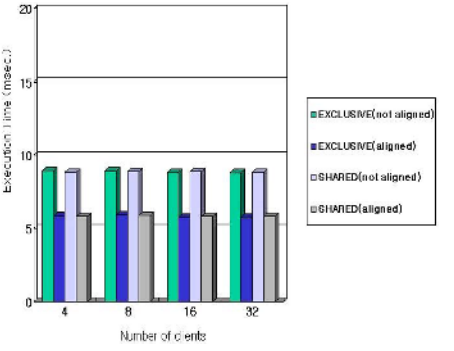


Fig. 6. Time overhead to acquire a distributed lock using four GLMs. Each client read from or wrote 0.8Mbytes of data to the distinct section of the same file.

4 Performance Evaluation

We measured the performance of the distributed locking protocol on the machines that have Pentium3 866MHz CPU, 256 MB of RAM, and 100Mbps of Fast Ethernet. The operating system installed on those machines was RedHat 9.0 with Linux kernel 2.4.20-8. The performance results focused on the time to obtain locks by performing lock revoke, downgrade, and upgrade operations. The times to invalidate client cached data and to write dirty data to disk were not included in the evaluation. In each experiment, we evaluated the effect of the lock range alignment. With the size of the lock range aligned, the data size being read from or written to storage is rounded up to the closest multiple of page size.

4.1 Evaluation of the Non-overlapped Data Range

Figures 5 and 6 represent, as the number of clients increases from 4 to 32, the time to obtain locks with the exclusive mode in write operations and with the shared mode in read operations.

When more than one machine were configured as GLMs, each lock request is given to a GLM, according to the round robin fashion. All clients read from or wrote 0.8Mbytes of data to the distinct portions of the same file. In this case, the lock requested by each client was newly created on GLM and returned to the requesting client, and thus caused no callback message to be sent to the remote lock holder.

As can be seen in Figures 5 and 6, with the size of the requested lock range aligned to page size, the better performance is obtained because, with the lock range alignment, each page can be assigned to exactly a single client at a time even though a slight larger data than actually requested must be read or

written. Without the alignment, boundary pages can simultaneously be accessed by neighbor clients, causing I/O serialization on those pages.

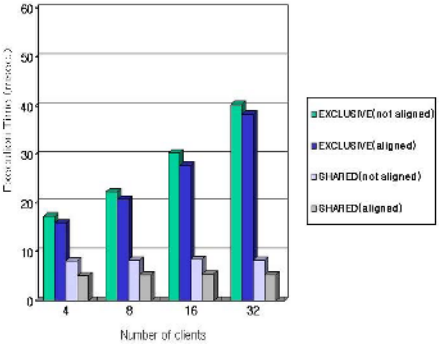


Fig. 7. Time to acquire a distributed lock using one GLM. A client's data range is shifted right so that each client is assigned the data section accessed by its neighbor at the previous step.

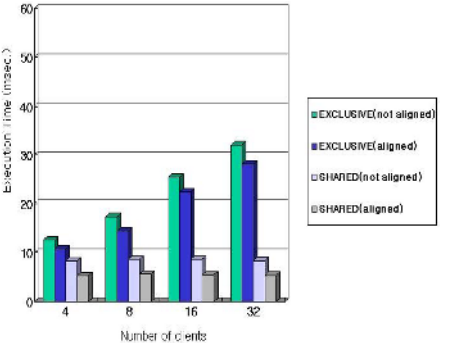


Fig. 8. Time to acquire a distributed lock using four GLMs. A client's data range is shifted right so that each client is assigned the data section accessed by its neighbor at the previous step.

4.2 Evaluation of the Overlapped Data Range

Figures 7 and 8 show the time to obtain locks with the exclusive mode and with the shared mode, while shifting right a client's data range so that each client is assigned the data section accessed by its neighbor at the previous step.

Figures 7 and 8 illustrate that the overhead of the lock revocation is significant with the exclusive mode because only a single client is allowed to write to a data section at any given time. With the shared mode, there is no need to contact the remote lock holder since a single lock can be shared between multiple nodes. With the shared lock mode, before granting a lock, GLM just increases a counter denoting the number of shared lock holders.

4.3 Evaluation of Lock Cost Per Node

Figures 9 and 10 show the elapse time to acquire locks, while changing the number of clients running on each node from 2 to 16. In order to fix the total number of clients to be 32, the number of nodes used in each experiment is 32 divided by the number of clients per node. For example, if 2 clients are running on a single node, 16 nodes are then used in the experiment.

In this experiment, with 2 clients running on the same node, the lock currently kept by a client would be used by its neighborhood at the next step. In other words, all the locks would locally be acquired every 2 I/O operations, without each client sending a callback message to the remote lock holder. On the other hand, with 16 clients, the callback message is sent to the remote lock holder every 16 I/O operations, reducing the cost for negotiating with the remote lock holders.

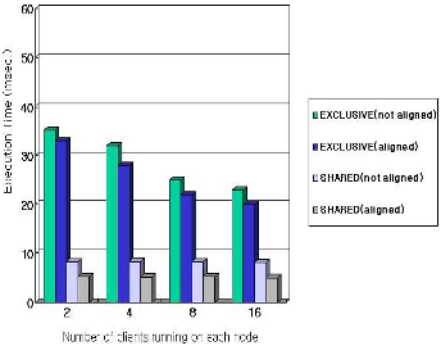


Fig. 9. Time overhead to acquire a distributed lock using one GLM as a function of number of clients running on each node. A client's data access range is shifted right at each step.

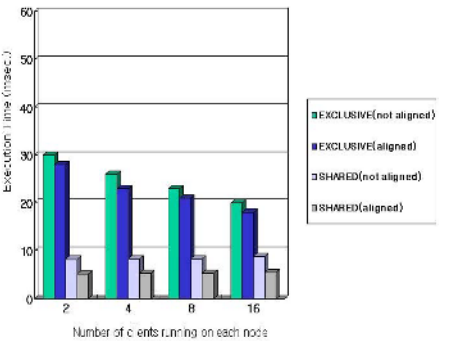


Fig. 10. Time overhead to acquire a distributed lock using four GLMs as a function of number of clients running on each node. A client's data access range is shifted right at each step.

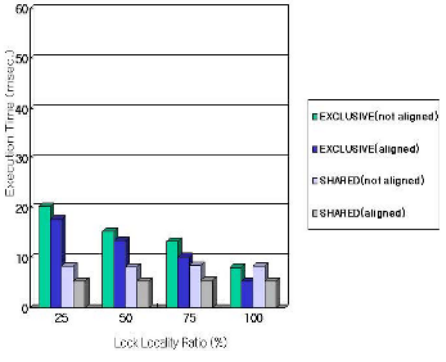


Fig. 11. Time to obtain a distributed lock as a function of lock locality ratio, using 16 clients with four GLMs. A client's data access range is shifted right at each step.

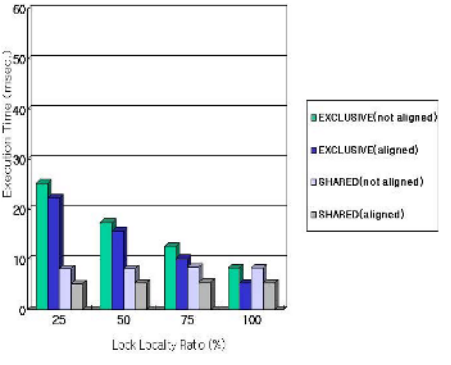


Fig. 12. Time to obtain a distributed lock as a function of lock locality ratio, using 32 clients with four GLMs. A client's data access range is shifted right at each step.

According to this experiment, we can observe that the dominating performance factor to acquire locks with 32 clients total is the network overhead to contact the remote lock holder.

4.4 Evaluation of Lock Locality

Figures 11 and 12 show the effect of the childlocks exploiting locality in the lock requests. Figure 11 shows the execution time taken to acquire locks using 16 clients with four GLMs and Figure 12 using 32 clients with four GLMs. Both experiments shown in

The lock locality ratio means how often childlocks are taken; if the lock locality ratio is of 25%, then 25% of total datalocks are childlocks needed to access to the smaller data portion than that of an already acquired datalock. If the lock locality ratio is of 100%, then all the locks are childlocks that do not cause communication overheads with GLM and with remote lock holders.

5 Conclusion

Concurrent accesses to the same file frequently occur in a distributed computing where allowing parallel write operations significantly improves I/O bandwidth. However, most distributed client-server file systems support a coarse-grained locking protocol in which all the concurrent write operations to a file are serialized even when the data sections being written are different between writers. In this paper, we presented a distributed locking protocol with which several nodes can simultaneously write to the distinct data portions of a file, while guaranteeing a consistent view of client cached data. The distributed locking protocol has also been designed to exploit locality of lock requests to minimize communication overheads with GLM and with remote lock holders.

References

1. Thekkath, C. A., Mann, T., Lee, E. K.: Frangipani: A Scalable Distributed File System, In Proceedings of the Symposium on Operating Systems Principles, 1997, pages 224–237
2. Preslan, K. W., Barry, A. P., Brassow, J. E., Erickson, G. M., Nygaard, E., Sabol, C. J., Soltis, S. R., Teigland, D. C., O'Keefe, M. T.: A 64-bit Shared Disk File System for Linux, In Proceedings of Sixteenth IEEE Mass Storage Systems Symposium Seventh NASA Goddard Conference on Mass Storage Systems & Technologies, March 15-18, 1999
3. Prost, J.-P., Treumann, R., Hedges, R., Jia, B., Koniges, A.: MPI-IO/GPFS, an Optimized Implementation of MPI-IO on top of GPFS, In Proceedings of Supercomputing, November 2001
4. Schmuck, F., Haskin, R.: GPFS: A Shared-Disk File System for Large Computing Clusters, In Proceedings of the First Conference on File and Storage Technologies(FAST), pages 231–244, Jan. 2002
5. Thakur, R., Gropp, W.: Improving the Performance of Collective Operations in MPICH, In Proceedings of the 10th European PVM/MPI Users' Group Conference (Euro PVM/MPI 2003), September 2003
6. Braam, P. J.: The Lustre stroage architecture, Technical Report available at - http://www.lustre.org, Lustre, 2002

Design and Evaluation of a Parallel Data Redistribution Component for TGrid

Sascha Hunold[1], Thomas Rauber[1], and Gudula Rünger[2]

[1] Department of Mathematics and Physics
University of Bayreuth, Germany
{hunold, rauber}@uni-bayreuth.de
[2] Department of Computer Science
Chemnitz University of Technology, Germany
ruenger@informatik.tu-chemnitz.de

Abstract. Data redistribution of parallel data representations has become an important factor of grid frameworks for scientific computing. Providing the developers with generalized interfaces for flexible parallel data redistribution is a major goal of this research. In this article we present the architecture and the implementation of the redistribution module of `TGrid`. `TGrid` is a grid-enabled runtime system for applications consisting of cooperating multiprocessor tasks (M-tasks). The data redistribution module enables `TGrid` components to transfer data structures to other components which may be located on the same local subnet or may be executed remotely. We show how the parallel data redistribution is designed to be flexible, extendible, scalable, and particularly easy-to-use. The article includes a detailed experimental analysis of the redistribution module by providing a comparison of throughputs which were measured for a large range of processors and for different interconnection networks.

1 Introduction

Heterogeneous distributed environments or grid environments provide large computation resources for the execution of extremely computation intensive scientific applications. These computing environments can be exploited to execute algorithms and applications with task-parallel structure. Those applications have to be transformed into modular component-based parallel programs which can be executed on a grid environment. A suitable programming model and an efficient runtime environment which implements the programming model are required for developing this kind of applications. The modular structure of programs can be expressed by a task graph. Each node in the task graph represents a single grid-enabled component. These components are implemented as multiprocessor tasks (M-tasks). A task graph containing M-tasks offers two levels of parallelism. Components can be executed concurrently but each component may also contain data-parallel code or may even contain a recursive structure of components.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 618–629, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The runtime environment TGrid is an approach for executing M-tasks in a heterogeneous distributed environment. The TGrid environment consists of several modules which enable the processing of M-tasks, the mapping of M-tasks to processors for execution, the observation of running M-tasks, and the redistribution of data between M-tasks [1]. M-tasks which can be executed in the TGrid environment are able to take full advantage of the underlying communication network by using MPI. On the other hand, since the M-tasks are written in Java, they are also completely platform independent. The redistribution of data structures within cooperating components plays an important role for component-based grid environments. There are several requirements that a data redistribution component has to meet to provide a solid framework for component-based programming on the grid. The data redistribution component must be flexible to support various data types and to be easily extendible if required. It should be simple to use, i.e. the developer should only provide all the information to allow an efficient data redistribution, and the component should support the coupling of components.

The contribution of this paper is the design and the implementation of a data redistribution module which enables the coupling of M-tasks in heterogenous computation systems. The redistribution module is able to dynamically create data messages from information provided by the source component (M-task) and to automatically redistribute these data onto the processors of the target component. In particular, the redistribution module relieves the program developer from writing redistribution code for each component, which is tedious and error-prone.

The rest of the paper is structured as follows. Section 2 gives a short introduction to the TGrid runtime system. In Section 3 the architecture of the redistribution module of TGrid outlined. Section 4 addresses the implementation details and introduces the management protocol of the redistribution module. Section 5 presents experimental results. Section 6 discusses related work, and Section 7 concludes the paper.

2 Overview of TGrid

TGrid is a runtime system for a network of heterogenous parallel machines which allows the execution of hierarchically-structured multi-processor tasks. Multi-processor tasks (M-tasks) can improve the performance of parallel programs due to a reduced communication overhead [2,3]. TGrid provides a framework to run these M-tasks on a heterogenous collection of clusters as proposed in [4]. TGrid consists of several subnets of which each is controlled by a subnet manager. The subnet manager is in charge of executing and observing tasks on its private network. Such a private network could be a homogeneous cluster or any other heterogenous collection of machines that share a private IP address space or are visible to each other. The subnet managers are able to transfer data through a WAN, which might be insecure. Therefore, the subnet managers support several protocols, such as https, ssh, etc., in order to bypass local firewalls and to perform encrypted data transfer.

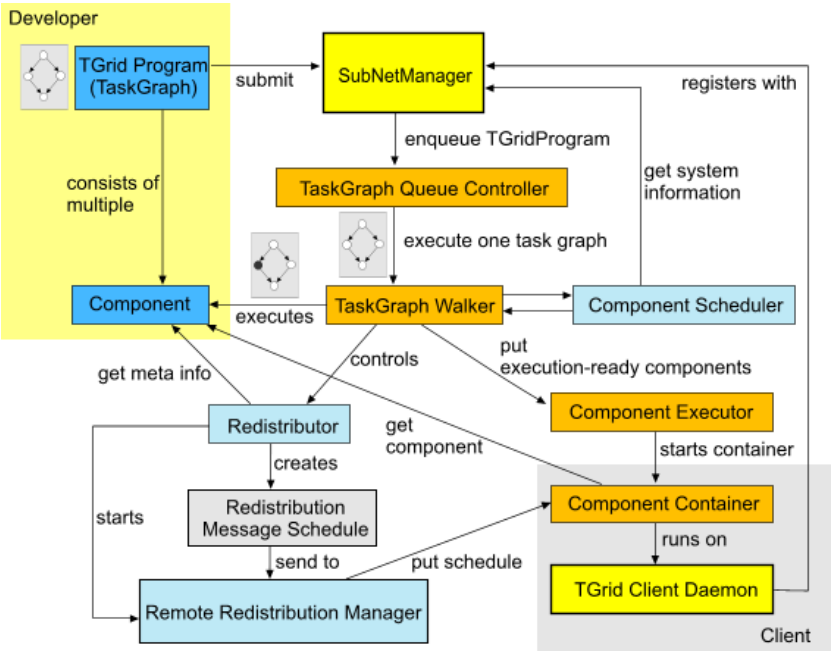


Fig. 1. TGrid architecture

The software architecture of the `TGrid` runtime system and an overview of the functioning of the subnet manager is sketched in Fig. 1. The program developer has access to one of the subnet managers and can submit programs to this local subnet manager. A `TGrid` program can be represented as a directed task graph where `TGrid` components are the nodes of the graph. Edges of the task graph represent dependencies between components, which includes data dependencies based on an output-input relation between components. If there is a data dependency between consecutive components A and B, such that B requires data structures produced by A, then the `TGrid` framework provides a redistribution component which can automatically satisfy the dependency (*coupling* of component). The subnet manager enqueues `TGrid` programs into a list of programs to be executed. The queue controller selects the next program to be executed on `TGrid`, spawns a TaskGraph Walker, and passes the user's program as parameter. The TaskGraph Walker executes the nodes of the task graph (components) as specified in the application program. An instance of a TaskGraph Walker observes the execution of a program throughout the entire life cycle of the program.

As soon as all input data dependencies of a node (component) are fulfilled a component is ready for execution. In `TGrid`, a component can only be executed within a single subnet of the grid runtime system. Components can be arbitrary single-processor or multi-processor tasks. `TGrid` is especially designed to support the execution of hierarchically-structured multi-processor tasks based on `TLib`

[2]. `TLib` is an MPI-based library that provides separate functions for the hierarchical structuring of processor groups and for the coordination of concurrent and nested M-tasks. `TGrid` components can also be executed on a remote subnet. If the component scheduler decides to execute a component in a remote subnet, the component will be sent to this target subnet and started remotely. As mentioned before, executing a component requires a component scheduling beforehand. The component scheduler maps a component to processors of a subnet taking into account the current workload of the subnet as well as the component's preferences and requirements, e.g. minimum and preferred number of processors to run this component. The scheduler can retrieve this information about the workload and the connected processors by calling interfaces provided by the subnet manager.

To eventually execute a scheduled component, the TaskGraph Walker starts a Component Executor which encapsulates the necessary actions to run the code. For instance, components using MPI have to be started differently than components using Java sockets. In case of MPI components, the client processor starts an MPI component container. An MPI component container initializes the MPI environment, registers itself with its local subnet manager, and waits for an MPI component to execute. The subnet manager sends the component to the component container. The component container checks if all data dependency of this component have been satisfied. If not, it notifies the subnet manager that it is ready for performing a data redistribution operation. When the data dependencies have been satisfied, the component container starts the execution of the component. The component container also informs the subnet manager when the component has finished its computation.

`TGrid` and all its components are written in Java which makes them completely platform independent. The current implementation supports components that are based on MPI. This approach allows us to run these components on arbitrary machines in the grid without having to recompile some parts. Moreover, the ability to access the MPI layer through JavaMPI enables the application to benefit from using the best network driver available on the corresponding machine.

3 Architecture of the MxN Redistribution Component

A parallel data redistribution, often known as MxN redistribution, is required for the coupling of programs (or components) which are executed in a data-parallel manner and have a data dependency. MxN stands for a data redistribution from M processors of the source program to N processors of the target program. Since all kinds of data redistributions between components require similar meta informations, the implementations follow similar patterns. The basic steps to redistribute data are data identification, message generation, message scheduling, and the actual communication. For `TGrid`, these basic steps are realized by the following tasks:

1. Specify the input and output *variables* of the redistribution. An output variable defines data which are provided by the sending component. Hence, input variables define data structures of the receiving component.

2. Define which parts of the source data has to be transferred to the target component (*selection*).
3. Define the data *mapping* between sending and receiving component.
4. Create a *schedule* of redistribution messages (communication schedule) that includes a sequence of messages that correctly moves data structures from the source to the target component.
5. Start communication and perform transfer.

In `TGrid`, the redistribution can be started when the source component has finished its computation and the target component has been initialized. The redistribution process has to determine all the meta information from both endpoints in order to create the communication schedule. Enumerating the information required to create a communication schedule may be straight-forward, however, it is difficult to design an easy-to-use and extendible (data-typing system) software component, i.e. `TGrid` provides generic data type interfaces which enables the developer to define and implement new data distribution types (e.g. block-cyclic 2-dimensional arrays of integers) if necessary.

A major concern is the throughput and the latency that can be provided in the communication stage. In order to gain good performance, we decided to differentiate between inter-subnet and intra-subnet redistribution. The data redistribution between two components which are part of the same subnet is referred to as intra-subnet redistribution. Intra-subnet redistribution in `TGrid` is characterized by direct message transfer from the sending processors to the receiving processors. In case that a component is executed by multiple operating system processes on one machine, direct message transfer to each of these participating processes is only possible if each processor can allocate arbitrary ports to receive data. Since most firewalls are configured to allow communication only on a few ports, we assume that unrestricted port management may only be allowed in private subnets. Unrestricted port management within a subnet enables the processors to directly transfer messages. As a result of the intra-subnet design, the message transfer within processors of the same subnet can be performed concurrently, i.e. no routing is involved. This ensures a high throughput and enables a fast redistribution within subnets. The communication between different subnets has different requirements. Many network configurations do now allow direct communication between a pair of processors, each located in a different subnet. However, firewalls may block traffic between two nodes and the IP address space may be completely private to a subnet. Instead, messages from one subnet to another have to be routed through the local subnet manager. The subnet manager has to be set up appropriately to support bypassing the firewall restrictions. The number of message hops increases when messages have to be routed over several other nodes which strongly influences the achievable latency and throughput.

Another important design goal for the redistribution module is to allow concurrent redistributions between components. The redistribution component of `TGrid` is implemented entirely in a multi-threaded way, i.e. different redistributions can be performed between each pair of components at the same time.

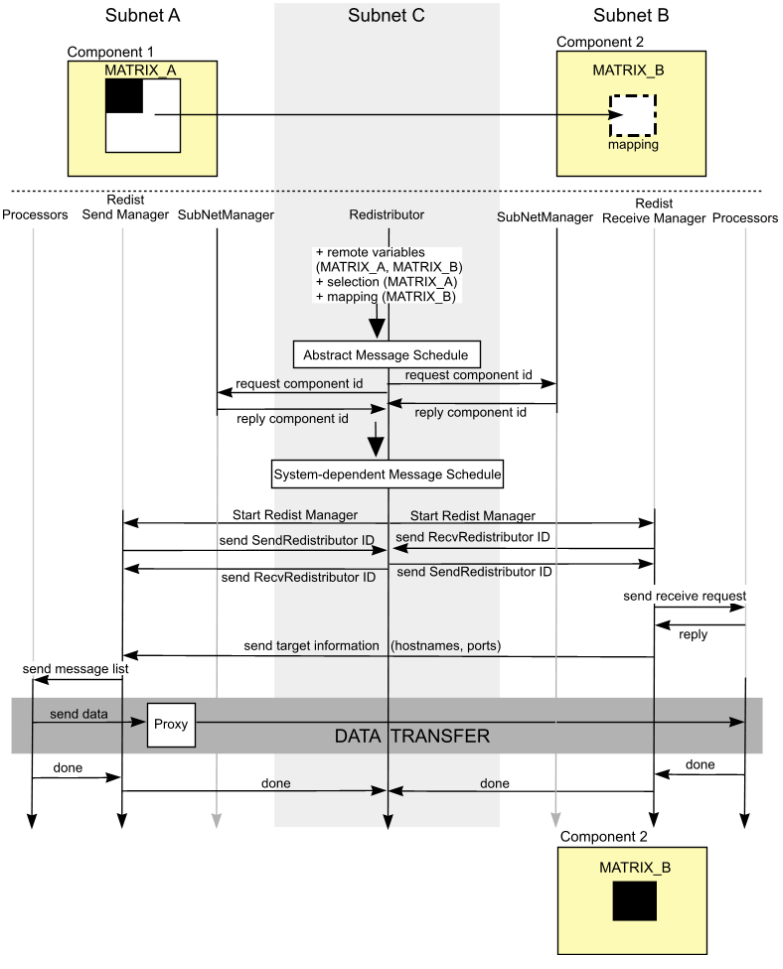


Fig. 2. `TGrid` Redistribution Protocol

4 Data Redistribution Protocol of TGrid

Data redistribution is realized by the `TGrid` Communication Protocol, which is shown in Fig. 2. A common case is that a sub-matrix computed by a component (Component 1) is required as input by another component (Component 2). The processors that execute these components may be located on different subnets or may be part of the same subnet. The redistributor is the entity that manages the entire redistribution and may be located on a third subnet, called Subnet C in Fig. 2. Since the redistributor is part of the execution of a `TGrid` program (see Fig. 1 in Sec. 2) it runs on the subnet where the user has submitted the application program. The user has to instruct the redistributor (by passing meta

information) which data has to be transferred between the components. This meta information includes

- *unique identifiers of variables* which determine the source and the target data structure in source and target component
- the *data selection* in the source component and the *data mapping* in the target component as described in Section 3.
- the *component schedule* of the source and the target component (number of processors, mapping). It enables the redistributor to retrieve information about the data layout which is required for generating the communication schedule.
- *references* to *source* and *target component objects*.

Using this information, the redistributor creates an abstract communication schedule. The abstract communication schedule is composed of a list of abstract messages which do not contain system information such as host names or IP addresses of the `TGrid` environment. In order to deliver messages in the current runtime environment, the redistributor has to convert the *abstract communication schedule* into a *system-dependent communication schedule*. Therefore, abstract messages are extended with a header which is used to uniquely identify the destination of a target variables. A component's variable can be uniquely identified within `TGrid` by specifying

- the *subnet* to uniquely identify the subnet to route the message to.
- the *component* to locate the component in the subnet. `TGrid` components get a unique label when they are passed to the runtime environment for execution.
- the *name of the variable* which stores the data to be accessed.

The *name of the variable* and the name of the *subnet* can be easily determined. The *component id* in `TGrid` however is assigned at runtime. To retrieve the component identifier from a remote subnet, the restributor sends a request to the remote subnet manager. The subnet manager responds by sending the corresponding *component id* back to the redistributor. The redistributor has now all information to convert the abstract communication schedule into a *system-dependent communication schedule*.

The redistributor starts redistribution managers on both subnets which manage the redistribution on the remote subnets and are primarily used to avoid communication across subnet border when it is not required. One redistribution manager controls the actions taken by the sending processors and the other controls the receiving processors, respectively. Redistribution managers belong to the same subnet as the sending or the receiving processors. Therefore, only data messages from source processors to target processors have to be sent across subnet borders. All other protocol messages that may be necessary for the redistribution, e.g. synchronization, require only communication within subnets (from processors to their redistribution manager).

`TGrid` is designed to run several redistributors and also redistribution managers concurrently. Therefore, each redistribution manager gets a unique ID

when instantiated. In order to determine the correct redistribution manager for incoming messages the redistribution managers have to exchange their IDs at first. Each redistribution manager sends its ID to the redistributor which forwards it to the remote redistribution manager.

The redistribution manager that controls the processors which have to send messages is referred to as *send manager* and the manager for the receiving processors as *receive manager*, respectively. The receive manager assigns a *unique port* to each receiving processor and sends a *receive request* to the processors. When all processors have responded to this request, the receive manager sends meta data of each receiving processor (*hostname*, *port*, *local rank*, etc.) to the send manager. The send manager uses this information to create a proxy object through which the sending processors send their messages. *Proxy objects* are used to make the *sending* of data messages *transparent* to the sending processor, i.e. the processor is not aware whether the target processor is part of the same local subnet or located on a remote subnet. With the information about the target host, the send manager can assign the message list and proxies to each processor that has data to contribute. Both the send manager and the receive manager wait until all cluster machines have acknowledged that the message transfer has been completed.

The sending or receiving of messages is not done by the component itself. This would force the component developer to implement redistribution code (sending/receiving) for every single component. Instead, the data transfer responsibility is part of the component container, see Section 2. Since component containers have no knowledge of the internal data structure of a component, the data transfer between a container and a component is realized by abstract interfaces which include methods to retrieve meta information (variable identifiers, data selection, data mapping) as discussed in this section.

5 Experimental Results

We performed a number of throughput experiments to evaluate the performance of the `TGrid` redistribution component. All intra-subnet tests were run on a cluster consisting of 64 Opteron processors (Model 246, 2 GHz). The cluster has three different interconnection networks per node; 100 MBit Ethernet, Gigabit Ethernet, and Melanox Infiniband. Currently, the `TGrid` framework provides a small number of distributed data types, e.g. distributed block-partitioned matrices, which play an important role in scientific applications.

The first experiment measures the throughput achieved when redistributing a block-partitioned matrix between two components in `TGrid`. Fig. 3(a) compares the throughput per node that was achieved with several matrix configurations and on different interconnection networks. The label 'mxn' denotes the number of processors that were assigned to the source (m) and to the target component (n). For instance, the label 'Gigabit 2x2' stands for the data redistribution over Gigabit Ethernet between 2 processors which run the source component and 2 processors which run the target component. As expected, the throughput per

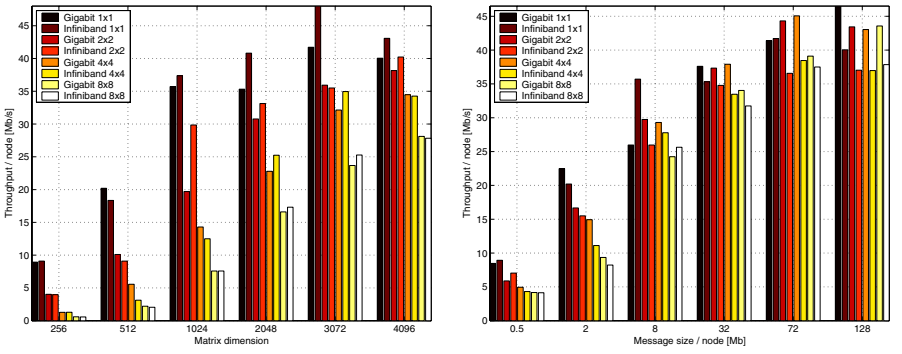


Fig. 3. (a) Intra-subnet data redistribution performance on Gigabit Ethernet and Infiniband. (b) Throughput comparison of several data redistributions on Gigabit Ethernet and on Infiniband. Constant message size per processor.

node increases with larger matrices. The message size that each processor has to send or receive depends on the size of the input matrix and on the number of processors that store the matrix. If more processors are used to store a matrix the throughput will be smaller. On the other hand, the protocol overhead has less impact on the overall time when the message size increases. For instance, for matrix size 4096 the throughput between 2 nodes ('1x1') is higher than measured with 16 nodes ('8x8'). The small throughput that can be seen for a matrix size of 256 is a direct result of the protocol overhead of the redistribution. The throughput of the redistribution is also limited by the object serialization done at the Java layer. The object serialization in Java is a very powerful tool and makes it easy to send object with different implementations across the network. These limitations cause the similar performance of Gigabit Ethernet and Infiniband.

Since the size of each message depends on the number of processors which store a distributed matrix, it is also important to examine the throughput when the size of matrices is adjusted in the way that the message size per processor is held constant. Fig. 3(b) shows the throughput per node with a constant message size per node (we used one processor per node on SMP boards). Again, the impact of the protocol overhead decreases the throughput for a small message size (0.5 MB) in comparison to bigger messages. We can also observe that the throughput per node is almost equal for a specific message size taking advantage of concurrent message redistribution.

Another important test for grid environments is to measure the bandwidth between long-distance networks. As mentioned above, `TGrid` is able to execute components remotely and can also handle the data redistribution between components which are located in different subnets. Fig. 4(a) compares the throughput achieved by inter-cluster redistribution (routed) with the throughput achieved by intra-cluster redistribution (direct). The inter-subnet redistribution tests were

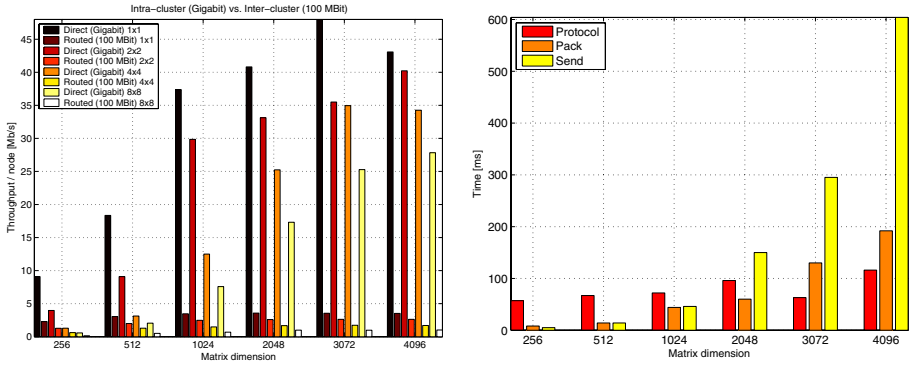


Fig. 4. (a) Intra vs. Inter-cluster redistribution; (b) Time spent in each redistribution step (matrix redistribution from 4 source processors to 4 target processors, 4x4, 8 nodes). Interconnection network: Gigabit Ethernet.

performed on a single cluster. To perform these tests, we divided the number of nodes into two disjoint subnets where each subnet is controlled by one node of the corresponding subnet. We also used another IP range with the consequence that the transfer between both subnets as well as the communication in between the subnet is done over the 100 MBit interconnection network. It is not surprising to see such a huge performance difference. Considering that the 100 Mbit card has a maximum bandwidth of 12.5 MB/s. Since each message is routed through two different subnet managers and since each message can only be forwarded to the next destination when it was entirely received, the throughput of a single message has an upper bound of $12.5/3 \approx 4$ MB/s (3 message hops to destination). Additionally, the sending of message from multiple processors has to be synchronized on a subnet manager to some extend which in turn reduces the possible throughput per node.

Fig. 4(b) shows the partial times spent in different redistribution steps. The bar labeled with 'Protocol' denotes the protocol overhead introduced by using the `TGrid` redistributor. The time 'Pack' denotes is the time for packing messages and the time 'Send' denotes the time for transferring the message over the network. The packing of messages contains determining the matrix elements to be sent, allocating a message buffer, selecting and copying elements from the local matrix buffer into the message buffer, and writing a message header containing meta informations for message reconstruction. We can observe that the protocol overhead is constant for different matrix sizes. For this reason, the `TGrid` protocol overhead has less impact on the performance for larger matrices. As we expected, the time for packing and sending messages increases linearly with the number of elements to be transferred.

The redistribution component of `TGrid` has shown good performance when taking into account that there is a price to pay for determining data, converting data, and managing arbitrary data inside a heterogenous grid environment.

6 Related Work

Data redistribution in distributed or grid environments has been an active area of research in the last couple of years. The Parallel Application WorkSpace (PAWS) provides a framework for coupling parallel applications within a component-like model [5]. The PAWS approach is similar to the redistribution component of `TGrid`, however, on another level of granularity. In contrast to PAWS, `TGrid` uses the notion of redistribution within an application, i.e. data redistribution is required to start a particular M-task of the application. The Model Coupling Toolkit (MCT) [6] is a software library written in Fortran 90 which provides functions to transfer data structures between parallel applications. InterComm [7] is a redistribution library that moves the determination of data redistribution patterns inside the programs to be coupled. An ongoing research project is carried out by the CCA (common component architecture) forum. The MxN working group of the CCA forum is working on the definition and implementation of interfaces to transfer data elements between parallel components running with different numbers of processes in each parallel component [8]. The framework Seine [9] is a geometry-based interaction model which is encapsulated as a CCA compliant component within the Ccaffeine CCA framework. Other CCA compliant frameworks that support coupling of distributed components are DCA [10] and XCAT [11]. An effective algorithm for communication schedule generation for data redistributions is presented in [12]. Messages between different clusters are scheduled in order to avoid exceeding the bandwidth capacity of the backbone.

7 Conclusions

In this article, we have presented the data redistribution component used by `TGrid`. The redistribution component is fully capable of managing arbitrary MxN redistributions. The redistribution framework of `TGrid` works completely multithreaded so that multiple variables of components can be transferred to arbitrary receiving components in parallel. The redistribution component provides simple but flexible interfaces which make it an easy task to extend the implemented data types and algorithms. The redistribution component differentiates between intra and inter-subnet communication. The intra-subnet communication allows the parallel sending of messages and is therefore of major importance for high throughput. The redistribution component of `TGrid` has shown good performance in the experiments.

References

1. Hunold, S., Rauber, T., Rünger, G.: TGrid – Grid Runtime Support for Hierarchically Structured Task-Parallel Programs. In: Proceedings of the Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, IEEE Computer Society Press (2006)

2. Rauber, T., Rünger, G.: Tlib - A Library to Support Programming with Hierarchical Multi-Processor Tasks. Journal of Parallel and Distributed Computing **65** (2005) 347–360
3. Hunold, S., Rauber, T., Rünger, G.: Multilevel Hierarchical Matrix Multiplication on Clusters. In: Proceedings of the 18th Annual ACM International Conference on Supercomputing, ICS'04. (2004) 136–145
4. Rauber, T., Rünger, G.: M-Task-Programming for Heterogeneous Systems and Grid Environments. In: Proc. of the IPDPS Joint Workshop on High-Performance Grid Computing and High-Level Parallel Programming Models, IEEE (2005)
5. Beckman, P.H., Fasel, P.K., Humphrey, W.F., Mniszewski, S.M.: Efficient Coupling of Parallel Applications Using PAWS. In: HPDC '98: Proceedings of the The Seventh IEEE International Symposium on High Performance Distributed Computing, Washington, DC, USA, IEEE Computer Society (1998) 215
6. Larson, J., Jacob, R., Ong, E.: The Model Coupling Toolkit: A New Fortran90 Toolkit for Building Multiphysics Parallel Coupled Models. Int. J. High Perform. Comput. Appl. **19** (2005) 277–292
7. Lee, J.Y., Sussman, A.: High Performance Communication between Parallel Programs. In: IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 4, Washington, DC, USA, IEEE Computer Society (2005) 177.2
8. Bertrand, F., Bramley, R., Damevski, K.B., Kohl, J.A., Bernholdt, D.E., Larson, J.W., Sussman, A.: Data Redistribution and Remote Method Invocation in Parallel Component Architectures. In: Proceedings of the 19th International Parallel and Distributed Processing Symposium: IPDPS 2005. (2005) Best Paper Award.
9. Zhang, L., Parashar, M.: Enabling efficient and flexible coupling of parallel scientific applications. In: International Parallel and Distributed Processing Symposium IPDPS'2006, Rhodes Island, Greece, IEEE Computer Society Press (2006)
10. Bertrand, F., Bramley, R.: DCA: A Distributed CCA Framework Based on MPI. In: 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico, USA, IEEE Computer Society (2004) 90–97
11. Krishnan, S., Gannon, D.: XCAT3: A Framework for CCA Components as OGSA Services. In: 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), CD-ROM / Abstracts Proceedings, 26-30 April 2004, Santa Fe, New Mexico, USA, IEEE Computer Society (2004) 90–97
12. Jeannot, E., Wagner, F.: Messages Scheduling for data Redistribution between Heterogeneous Clusters. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2005). (2005)

MetaLoRaS: A Predictable MetaScheduler for Non-dedicated Multiclusters*

J. Ll Lérida[1], F. Solsona[1], F. Giné[1], M. Hanzich[2], P. Hernández[2], and E. Luque[2]

[1] Departamento de Informática e Ingeniería Industrial, Universitat de Lleida, Spain
{jlerida, francesc, sisco}@diei.udl.es
[2] Departamento de Arquitectura y Sistemas Operativos, Universitat Autònoma de Barcelona, Spain
mauricio@aomail.uab.es, {porfidio.hernandez, emilio.luque}@uab.es

Abstract. The main aim of this work is to take advantage of the computer resources of a single organization or Multicluster system to execute distributed applications efficiently without excessively damaging the local users. In doing so we propose a Metascheduler environment named MetaLoRaS with a 2-level hierarchical architecture for a non-dedicated Multicluster with job prediction capabilities.

Among other Metaschedulers, the non-dedicated feature and an efficient prediction system are the most distinctive characteristics of MetaLoRaS. Another important contribution is the effective cluster selection mechanism, based on the prediction system.

In this article, we show how the hierarchical architecture and simulation mechanism are the best choices if we want to obtain an accurate prediction system and, at the same time, the best turnaround times for distributed jobs without damaging local user performance.

1 Introduction

In this paper, we consider the issue of metascheduling jobs across a Multicluster. A Multicluster system has a network topology made up of interconnected clusters and limited to a campus- or organization-wide network with a common domain name, while a traditional grid is national or even global in scale [6]. Instead of grid computing, Multicluster systems allow the construction of efficient prediction systems based on the availability and reliability of their constituent resources.

Several studies [2] have revealed that a high percentage of computing resources in a Network Of Workstations (NOW/Cluster) are idle. The possibility of using this computing power to execute parallel jobs with a performance equivalent to a Massively Parallel Processor (MPP) and without perturbing the performance of the local user applications on each workstation has led to a proposal for new resource management environments ([7,11]).

With the aim of taking advantage of these idle computing resources available across the cluster, in previous work [4,5], we developed a cluster scheduling system named LoRaS (Long Range Scheduler). The most important feature of LoRaS is its efficient

* This work was supported by the MEyC-Spain under contract TIN 2004-03388.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 630–641, 2006.
© Springer-Verlag Berlin Heidelberg 2006

turnaround predictor for the parallel jobs, which allows the most efficient space sharing scheduling policy to be selected depending on the cluster state. In this article, we present MetaLoRaS, an extension of LoRaS, which provides the ability for a non-dedicated Multicluster to act as a Metascheduler. MetaLoRaS deals with each cluster by means of the LoRaS scheduling system requesting the cluster state and the prediction of the job turnaround time.

MetaLoRaS has a two-level hierarchical architecture. This choice is based on the work performed in [3], where an evaluation of the interaction between Metaschedulers and local schedulers (in each cluster) is presented. One important conclusion they presented was that among all the strategies considered, the best is to have one scheduler in each cluster, and to drop the requirement for scheduling single-component jobs to the cluster (or bottom) level.

The major scheduling decision of the top level of MetaLoRaS is to select the cluster to which the parallel jobs are delivered for execution. MetaLoRaS decisions are based on minimizing the execution time of the jobs without adding an excessive overhead to the local tasks. Parallel job execution times in each cluster are predicted in their respective LoRaS system, the bottom level of MetaLoRaS.

Most cluster selection policies used by Metaschedulers in the literature, employ traditional First Fit, Best Fit, or Greedy scheduling, or some variant on these ([13,10,9,3]). Moreover, an integral space sharing scheduling with load redistribution between clusters was used in [1]. A more sophisticated genetic algorithm was presented in [8]. An alternative technique to distribute the jobs between the clusters was proposed in [12] and [9]. They proposed distributing each job to the least loaded clusters in a redundant manner, thus improving the system utilization and reducing average job slowdown and turnaround time. However, the overhead involved in implementing this was found to be a problem.

Although these above-explained environments can give acceptable performance, they are not envisaged for non-dedicated environments and they cannot be used to predict job execution times in non-dedicated Multiclusters, the most important aims of this work.

In this article our metascheduling system was tested in a real Multicluster made up of three clusters. The results obtained demonstrate the good behavior of our Metascheduling mechanism and its applicability to Multicluster systems. Special attention was paid to the main contribution of this article, the cluster scheduling policy, that is, the algorithm which assigns jobs to clusters.

The remainder of this paper is outlined as follows. In Section 2, the Multicluster Scheduling system (MetaLoRaS) is presented. The efficiency measurements of MetaLoRaS are performed in Section 3. Finally, the main conclusions and future work are explained in Section 4.

2 MetaLoRaS

MetaLoRaS is based on a previously developed cluster scheduling system named LoRaS. LoRaS is a space sharing scheduler which allows a large number of scheduling facilities to be defined.

In this section the main features of LoRaS are introduced. Next, the Multicluster architecture is extensively explained. In doing so, the main components of MetaLoRaS and their functionality are defined. Finally, the Metascheduling mechanism, consisting of selecting the cluster where the jobs are mapped, is explained separately.

2.1 LoRaS

LoRaS ([4,5]) is basically a Queue Manager developed in order to provide a space sharing scheduling facility for non-dedicated cluster systems. As can be seen in Fig. 1, the most important components of LoRaS are the Input Queue, the Predictor and the Scheduler.

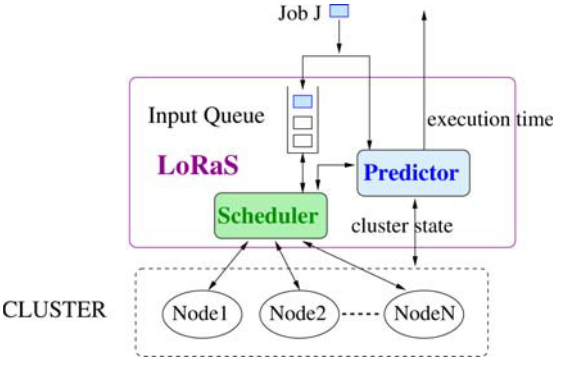


Fig. 1. LoRaS

The jobs in the Input Queue are ordered in a FCFS manner. The mapping policy of LoRaS selects the nodes for executing a parallel application considering the level of resource usage throughout the cluster. Thus, it does not overload any node without damaging the local user interactiveness.

LoRaS provides a means for predicting the execution time of parallel jobs in each cluster. This prediction is performed by simulation. For our purpose, the simulator is the most important component of LoRaS, because it is the basis for the Metascheduling mechanism proposed in this article.

In order to deal with this estimation, LoRaS uses information about the parallel applications and the cluster state. Each parallel application is executed in isolation and the following information is gathered: number of requested nodes, execution time and percentage of used CPU and Memory. The cluster state is modeled according to the following information about each cluster node: average load, memory occupancy, local user activity and the Multi-Programming Level (MPL), which is the number of parallel applications running in the node. If LoRaS sets the maximum MPL as N, no more than N parallel tasks can be executed in each cluster node.

LoRaS also deals with user activity. The nodes where there are local user activity, the MPL is adjusted accordingly. This way the local user jobs are also considered when

assigning new parallel tasks. The Metascheduler proposed in the next section improves the treatment of the local applications significantly.

As was demonstrated in [4] and [5], taking into account the information described above, the predictor can obtain accurate estimates of the turnaround time of the distributed jobs. In fact, the turnaround deviation achieved by the predictor system is always below 20%.

2.2 MetaLoRaS Architecture

First of all, we must decide whether the Multicluster system architecture will be centralized, hierarchical or distributed. Multiclusters are usually based on high-speed local networks, and therefore a centralized or hierarchical scheme should be suitable.

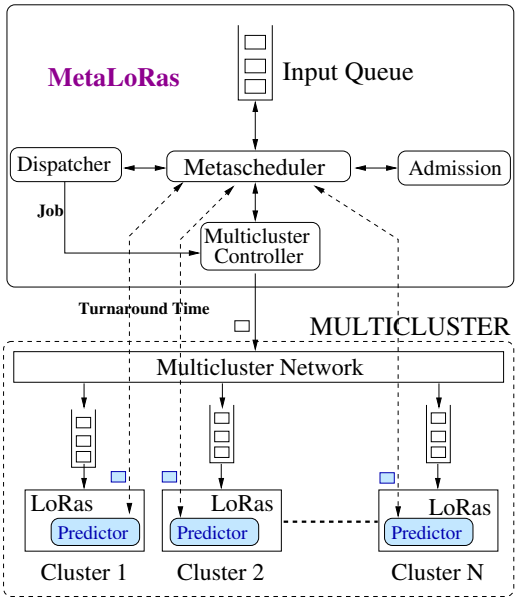


Fig. 2. MetaLoRaS

We are interested in extending LoRaS to make a Metascheduler system oriented to non-dedicated Multiclusters. The final result is MetaLoRaS. Unlike a centralized scheme, where the scheduling process should be performed in the Metascheduler node, we chose the hierarchical model. This decision was based on the attempt to split metascheduling decisions between the overall cluster or more precisely between the LoRaS front-end nodes, thus reducing the added scheduling overhead in the system. Furthermore, as was pointed out in [3], the best results were obtained when there was one scheduler in each cluster.

MetaLoRaS is made up of five components (see Fig. 2): the Input Queue (a queuing system), the Multicluster scheduler (named Metascheduler), the Admission system, the Dispatcher and the Multicluster Controller.

It is reasonably expected that users outside the organization should not have direct access to the local resources. Instead, they should use the Multicluster through one entry point, the Input Queue. Note that the Input Queue should comprise more than one queue. Doing so, the jobs could be classified, for example, on the basis of their scheduling priority. Nevertheless, for simplicity reasons, only one Input Queue is considered in this article.

MetaScheduler, the MetaLoRaS scheduler, is the Multicluster scheduling system. It is responsible for selecting the next job to be executed from the Input Queue, and also the cluster where this job will be executed. *Metascheduler* is explained in depth in section 2.3.

The *Admission System* is responsible for admitting new jobs into the system. This module will accept the new job whenever its required resources fit on at least one cluster. If not, the job is discarded. The specified resources are the number of processors and the size of Memory.

The MetaLoRaS maintains an updated list of the state of each cluster, so it can determine in real time whether the first job in the Input Queue can be executed. This information is obtained from the Multicluster Controller system. The *Multicluster Controller* collects real time information about the state of each cluster. If an event occurs in one cluster (job start, finish), the Multicluster Controller is notified of such a change. It is responsible for updating the database with the information from the different clusters. In doing so, it collects the changes performed in each cluster. The LoRaS system is responsible for notifying the Multicluster Controller about the cluster state changes.

The *Dispatcher* has the ability to send the job to the Input Queue of the LoRaS frontend, selected by MetaLoRaS. The jobs to be executed can be launched in both PVM and MPI formats.

One important issue to be taken into account is the Multicluster Network topology. The nodes are usually grouped into clusters because they share an internal network. A Multicluster has a distinctive architectural feature: the internal networks of the clusters are bridged together through dedicated links. This configuration increases the complexity of effectively managing both computing and networking resources.

Finally, we suppose that the clusters making up the Multicluster are accessible from the node where the MetaLoRaS is located. Also, we suppose that the clusters are shared between local user applications and the parallel jobs.

2.3 MetaLoRaS Scheduling System

In this section the functioning of MetaScheduler, the Multicluster scheduling system (see Algorithm 1) is explained. MetaScheduler is responsible for selecting jobs from the Input Queue and mapping them into clusters, controlled by the LoRaS system.

The next job (J) from the Input Queue to be executed is selected in a FIFO manner. In this article, as we are interested in presenting an efficient prediction system, only the FIFO policy is considered because optimal prediction accuracy is obtained when using this policy [5]. If the selected job J cannot be executed in the Multicluster, i.e. the resources that the job requires do not fit in one cluster, the admission system will reject the job.

Each job must provide the number of processors and their respective CPU and Memory requirements. As mentioned in the previous section, they are obtained by the LoRaS when executing in isolation.

Algorithm 1. Cluster Selection Policy.

Step 1 Choose in a FIFO manner the next job (J) from the Input Queue
Step 2 If there are not enough resources in every Cluster, reject job J and **Go to** Step 1
Step 3 Obtain the Cluster C to map job J according to the Prediction Algorithm (Algorithm 2).
Step 4 Dispatch Job J to Cluster C
Step 5 **Go to** Step 1

If the selected job J is not rejected (fits in almost one cluster), it will be mapped into one cluster by means of the Cluster Selection Policy (see algorithm 1). Jobs are selected in a FIFO manner and the considered resources are the Memory occupancy and the maximum fixed MPL of the cluster nodes. This information is maintained by the Multicluster Controller system. For this reason, each cluster must provide information to the Multicluster Controller when jobs complete and free processors.

The proposed policy is based on simulation. The job execution is simulated in each LoRaS simulator, residing in the front-end node of each cluster. With this model, the clusters do not perform any scheduling decision, but are only responsible for dispatching the jobs that are supplied by the MetaLoRaS. The *Prediction Algorithm* (Algorithm 2) summarizes the steps performed by the simulation process in obtaining the cluster to map the job J.

Next, the *Prediction Algorithm* is explained. The turnaround time of a job J is obtained by simulation in all the clusters. In doing so, the cluster state, the waiting jobs in the LoRaS Input Queue and the local user activity is taken into account. To select the cluster, the MetaLoRaS is based on the Pondered Turnaround Time metric ($PTT(J)$).

Let C be a Multicluster made up of "n" clusters. The Pondered Turnaround Time for a cluster k$\leq$n ($\mathrm{PTT_k}$) and a job J, is defined as:

$$PTT_k(J) = TR_k(J)\left(W\frac{TR_k(J)}{MaxTR(J)} + (1-W)\frac{LT_k}{MaxLT}\right), \quad (1)$$

where $TR_k(J)$ is the predicted turnaround time obtained in cluster k, $MaxTR(J)$ is the maximum predicted turnaround time across the Multicluster, LT_k is the number of nodes assigned to job J in cluster k with local user activity and finally, $MaxLT$ is the maximum number of nodes assigned to job J with local activity across the Multicluster. To prevent a division by 0, $MaxLT$ is set to 1 when there is no local activity at all ($MaxLT$=0). W (a real value in the range [0..1]) is the weight assigned through the formula to the turnaround time. 1 - W will then be the weight assigned to the resource occupancy of user applications.

We define the Pondered Turnaround Time of a job J as the minimum $PTT_k \; \forall \, \mathrm{k} = 1..n$. This is:

$$PTT(J) = min_{k=1}^{n}\{PTT_k(J)\} \quad (2)$$

Our Metascheduling system will assign the job J the cluster which gives the minimum turnaround time and that disturbs the local user applications as little as possible. According to equations 1 and 2, Cluster k with $PTT(J) = min_{k=1}^{n}\{PTT_k(J)\}$ will be selected.

Algorithm 2. Prediction Algorithm.

Step 1 Metascheduler obtains the list of clusters (L_c) where the job can be executed.
Step 2 For each cluster $C \in L_c$, LoRaS obtains its cluster state.
Step 3 Metascheduler request the turnaround time of Job J in each cluster k where the job can be executed ($TR_k(J)$). Also request the local resource occupancy of each cluster (LT_k, $\forall k = 1..n$).
Step 4 Metascheduler computes $PTT_k(J)$, $\forall k = 1..n$.
Step 5 Metascheduler obtains the cluster C where $PTT(J) = min_{k=1}^{n}\{PTT_k(J)\}$ is reached.

By varying W we can obtain different scheduling capabilities. If the local user applications must not be delayed by the parallel jobs, we will set $W \simeq 0$. Instead, If the major performance metric to be taken into account is the turnaround time of parallel applications, a $W \simeq 1$ should be used. For a value of $W \simeq 0.5$, equal importance will have both local user intrusion and parallel turnaround times.

Note that there can also be different W values for different cluster systems. This should be used for example to distinguish between clusters where the user applications cannot be damaged at all ($W \backsimeq 0$).

3 Experimentation

In order to carry out the experimentation process, we need the user and parallel workloads. The user workload is represented by a synthetic benchmark (named *local_bench*) which can emulate the usage of 3 different resources: CPU, Memory and Network traffic. The use of these resources was parametrized in a real way. According to the values obtained by collecting for a couple of weeks the user activity in an open laboratory, *local_bench* has been modeled to use 15% CPU, 35% Memory and 0,5KB/sec LAN.

The parallel workload was a list of 60 NAS parallel jobs (CG, IS, MG, BT), with a size of 2, 4 or 5 tasks, that reached the system following a Poisson distribution. These jobs were merged so that the entire parallel workload had a balanced requirement for computation and communication. It is important to mention that the parallel MultiProgramming Level (MPL) reached for the workload depends on the system state at each moment, but cannot exceed an $MPL = 4$.

In this experimentation we are interested in knowing the benefits of the new *Cluster Selection Policy* (named PTT) based on the state of the Multicluster. In order to be able to show the benefits of this policy, a comparison was done between different policies with different values of local activity. The policies compared are: FFIT (First Fit), RR (Round Robin), TAT (Turnaround Time) and PTT (Pondered Turnaround Time). The first two are classic policies, whereas TAT and PTT use the LoRaS prediction system. TAT assigns jobs in the cluster with the minimum turnaround time. In PTT (the policy

presented in this article), not only is the minimum turnaround time taken into account (as in TAT), but also low user intrusion is considered, as was expressed in Formula 2.

The whole system was evaluated in a Multicluster made up of three non-dedicated clusters with the same number of nodes. The 3 clusters, named Cluster1, Cluster2 and Cluster3 each had 5. Each cluster comprised 3GHz workstations with 1GB of RAM and 2048 KB of cache, interconnected by a 1Gigabit network.

3.1 Metrics

In order to be able to compare the different policies three different metrics, *Turnaround*, *Makespan* and *Slowdown*, were used.

The *Turnaround* measures the mean elapsed time in executing one job of the parallel workload. This metric allows us to measure the efficiency of the different Cluster Selection Policies evaluated from the parallel application point of view.

Makespan, on the other hand, allows us to analyze the global performance of the overall system. It gives the elapsed time from the start of the first job to the end of the last job. This metric relates the performance of the system in executing the overall parallel workload. It is considered a system centric metric.

To quantify the intrusion introduced by the execution of the parallel workload in the user applications, we used a benchmark, called *local_bench*, that measures the averaged latency of a set of system calls. The *Slowdown* (*SL*) of local user applications is defined as follows:

$$SL = \frac{AvLat_{non-dedicated}}{AvLat_{dedicated}} \tag{3}$$

where $AvLat_{non-dedicated}$ is the averaged latency value in executing *local_bench* with the parallel workload (non-dedicated environment) and $AvLat_{dedicated}$ is the averaged latency of the same benchmark when executed in isolation (dedicated environment).

3.2 Performance Comparison

In the Figure 3 we show the averaged Turnaround time of the parallel workload by varying the local activity. 0% means that there are no local users in any cluster; 30% one cluster is entirely occupied by user applications; 60% two clusters are occupied and finally 100%, the overall Multicluster has user activity.

As can be seen in the figure, the FFIT policy is the worst. In this case, almost all the parallel jobs were assigned to the same cluster (the first one) because there were enough resources. Only when there were not enough processors to execute the job was another cluster selected.

The RR policy behaved better than FFIT in all the situations, because it distributed better the jobs. As Figure 3 shows, the policies TAT and PTT mapped even better because the assignment is based on the prediction of the turnaround time. This improvement also demonstrates the effectiveness of the prediction module of LoRaS.

As we can see, the behavior of the PTT model depends on the value of W and the user workload. PTT with W=1 gave similar performance to TAT in all the cases. As Formula 2 shows, this result was expected because in both cases only the turnaround time is considered.

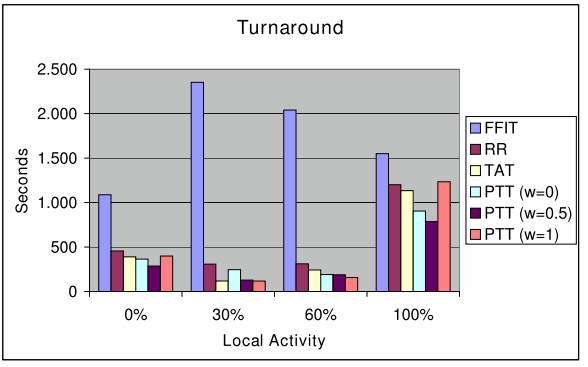


Fig. 3. Turnaround

For the W=0 case, the PTT model discards the clusters with local activity. When there are various clusters without user activity, the policy selects the first cluster with enough resources. In many situations the assignments are performed in the same cluster, penalizing the Turnaround metric, as passed in the 30% case.

The best performance results are obtained for PTT with W=0.5 and W=1. This is due to the fact that in both cases the turnaround time is considered, avoiding the commented problems of W=0. The average of the performance results is similar in both cases.

PTT increased the performance of RR and FFIT in all the situations. Also, on average, the PTT model with W=0.5 and W=1 improved TAT in 5%.

The Figure 4 shows the Makespan results. This metric measures the overall system performance with respect to the applied policy. As was expected, the obtained results are very similar to the turnaround. This fact corroborates the good behavior of our experimental environment and the implemented policies.

With regard to the Makespan metric, PTT is also the best policy. Obtaining, in average, a gain with respect to TAT of 2% for PTT W=0 and 8% for PTT with W=0.5 and W=1.

3.3 User Slowdown

In this section, the influence of the different policies on the user applications is evaluated.

Figure 5 shows the Slowdown obtained when executing *local_bench* in the same situations as in the previous section. As can be seen, the intrusion does not increase with the local activity. This demonstrates that LoRaS deals with the local user well, by balancing the maximum MPL allowed in the cluster with the local activity. The greater the local activity, the smaller the MPL and this causes that the latency to stay stable.

In general, a great negative impact on the local task performance was produced by the FFIT model. In this case, only taking into account the fitting of the jobs in the assignment increased the intrusion in the user applications. The RR model also does so. With a simple distribution of the load it also caused a considerable overhead in the user workload.

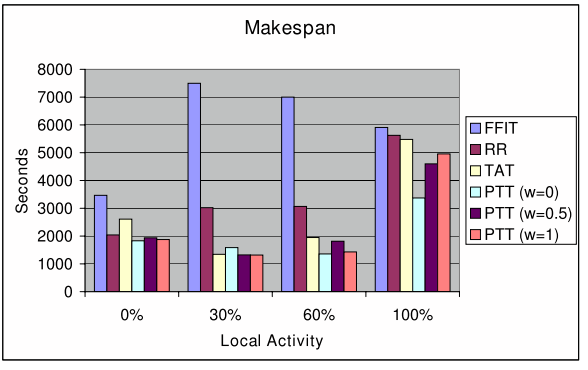


Fig. 4. Makespan

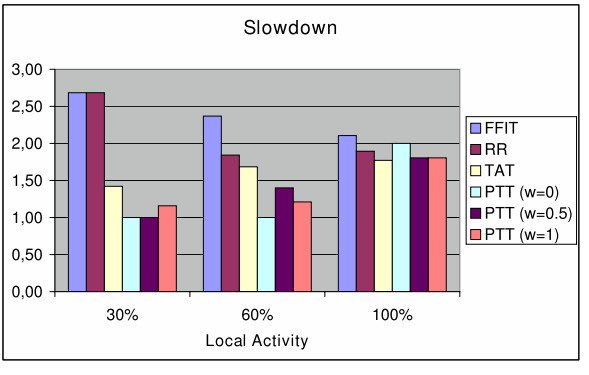


Fig. 5. Local Intrusion

We can observe how PTT maintains the smallest latency in almost all the cases. The exception occurred in the case of W=0 and 100% user activity. This was produced because this model tries to decide only according to the user activity. As all the clusters have the same user workload, this model decides in arbitrarily.

We have verified that in average, the PTT model with W=0 obtained the smallest latency. However, the TAT model improved performance of parallel applications with respect to PTT with W=0 in 11%. We have also verified that the best performance is obtained for PTT with W=0.5 and W=1, with a gain with respect to TAT of 5%. Furthermore, PTT with W=1 obtained smaller latency values than PTT with W=0.5.

4 Conclusions and Future Work

In this article we have presented MetaLoRaS, a Metascheduler system based on simulation with a two-level hierarchical architecture for a Multicluster environment. Compared with traditional policies in the literature, the simulation model we propose gave the best cluster selection scheduling because it can predict the turnaround time of an

application in a particular cluster system. This way, no redundant job assignments must be implemented to obtain the best result (as in [12]).

The proposed Metascheduler gives good parallel performance without excessively damaging the user workload. This is one of the most important aim of this work. Our Metascheduler is very efficient when it is based only on the prediction mechanism. It reduced the Makespan of the parallel workload by 23% and the turnaround by 25% with respect to the classic policies (First Fit and Round Robin).

The proposed new policy, PTT, based on the prediction mechanism and the user activity allows the intrusion of user applications to be reduced by an average of 9%. Varying the W factor, we obtain different performance results depending on the user activity. For the W=0 case, we obtained better *Slowdown* results. Instead, for the W=0.5 model, we obtained better average turnaround and Makespan values. W=1 obtained gains similar to W=0.5 but reducing the user intrusion. W=1 is the best mode if we want to balance parallel against user application performance. To improve the overall system performance with low user intrusion, W=0 should be chosen. Instead, W=0.5 is the best model if the goal is to improve the parallel application performance.

Future work is directed towards improving the scheduling model when the user activity is similar across the Multicluster. As our model only considers the presence of user activity, it do not select efficiently the clusters to map the jobs when the clusters are evenly loaded. We need additional information, such as CPU or Memory occupancy to distinguish the least loaded cluster to perform more efficient decisions. We want also consider the workload changes in advance, making load-redistribution (as in [1,9]) unnecessary.

The number of jobs waiting in the Input Queue of the LoRaS systems reduce prediction accuracy. To correct this problem new proposals in the PTT policy should be made.

References

1. J. Abawajy and S. Dandamudi. Parallel Job Scheduling on Multicluster Computing Systems. In Proceedings of the IEEE International Conference on Cluster Computing (CLUSTER'03). 2003.
2. A. Acharya and S. Setia. Availability and utility of idle memory in workstation clusters. In Proceedings of the ACM SIGM./PERF.'99, pp. 35–46. 1999.
3. A. Bucur and D. Epema. Local versus Global Schedulers with Processor Co-allocation in Multicluster Systems. JSSPP 2002, LNCS 2537, pp. 184–204. 2002.
4. M. Hanzich, F. Giné, P. Hernández, F. Solsona and E. Luque. Coscheduling and Multiprogramming level in a non-dedicated cluster. LNCS vol. 3241, pp. 327–336. 2004.
5. M. Hanzich, F. Giné, P. Hernández, F. Solsona and E. Luque. Cisne: A new integral approach for scheduling parallel applications on non-dedicated clusters. LNCS vol. 3648, pp. 220–230. 2005.
6. W. Jones, W. Ligon, L. Pang. Characterization of Bandwidth-Aware Meta-Schedulers for Co-Allocating Jobs Across Multiple Clusters. The Journal of Supercomputing, vol. 34, pp. 135–163. 2005.
7. M. Litzkow, M. Livny, and M. Mutka. Condor- a hunter of idle workstations. 8th Int'l Conference of Distributed Computing Systems. 1988.

8. A. Bose, B.Wickman and C. Wood. MARS: a metascheduler for distributed resources in campus grids. Proceedings. Fifth IEEE/ACM International Workshop on Grid Computing. 2004.
9. G. Sabin, R. Kettimuthu, A. Rajan and P. Sadayappan. Scheduling of Parallel Jobs in a Heterogeneous Multi-site Environment. JSSPP 2003, LNCS 2862, pp. 87–104. 2003.
10. J. Santoso, G.D. van Albada, B.A. Nazief, P.M. Sloot. Hierarchical Job Scheduling for Clusters of Workstations. Proceedings of the sixth annual conference of the Advanced School for Computing and Imaging (ASCI 2000), pp. 99-105. 2000.
11. E. Shmueli and D. G. Feitelson. Backlling with lookahead to optimize the performance of parallel job scheduling. Job Scheduling Strategies for Parallel Processing, LNCS, 2862:228251. 2003.
12. V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing. 2002.
13. M. Xu, "Effective Metacomputing using LSF MultiCluster," ccgrid, p. 100, 1st International Symposium on Cluster Computing and the Grid, 2001.

The Cost of Transparency: Grid-Based File Access on the Avaki Data Grid

H. Howie Huang and Andrew S. Grimshaw

Department of Computer Science, University of Virginia
151 Engineer's Way, Charlottesville, Virginia 22904
{huang, grimshaw}@cs.virginia.edu

Abstract. Grid computing has been a hot topic for a decade. Several systems have been developed. Despite almost a decade of research and tens of millions of dollars spent, uptake of grid technology has been slow. Most deployed grids are based on a toolkit approach that requires significant software modification or development. An operating system technique used for over 30 years has been to reduce application complexity by providing transparency, e.g. file systems mask details of devices, virtual machines mask finite memory, etc. It has been argued that providing transparency in a grid environment is too costly in terms of performance. This paper examines that question in the context of data grids by measuring the performance of a commercially available data grid product – the Avaki Data Grid (ADG). We present the architecture of the ADG, describe our experimental setup, and provide performance results, comparing the ADG to a native NFS V3 implementation for both local and wide area access cases. The results were mixed, though encouraging. For single client local file operations, native NFS outperformed the ADG by 15% to 45% for smaller files, though for files larger than 32 MB ADG outperformed native NFS.

1 Introduction

Grid computing has become a popular and much-hyped term in both academia and industry, has been the subject of numerous research projects, and has been embraced as a key technology by major software vendors such as IBM, HP, Oracle, Sun, and others. Today, grid systems are still an active area of computer research while at the same time software based on grid technology has spread into production use in industries as diverse as pharmaceuticals, aerospace, financial services, and telecommunications, for sharing enterprise resources [1-3].

Grid computing is a much misunderstood concept. Many believe that it is primarily or even solely concerned with high performance execution environments – i.e. ones dedicated to improve the performance of high performance/parallel applications such as large individual MPI applications, or high throughput applications such as SETI@home and parameter space studies. However, the full scope and potential of grid computing is much greater than simply coupling together CPUs to provide more collective cycles. Rather, it is a concept that focuses system design on providing secure sharing and easy, virtualized access to resources of many

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 642 – 659, 2006.
© Springer-Verlag Berlin Heidelberg 2006

types, e.g. CPU, storage, data, applications, policies, instruments, etc., across wide-area networks and across organizational boundaries [4-7].

While the majority of the early research and development of grid systems focused primarily on developing advanced execution environments [8-11], a growing number in the grid community have realized that it is desirable – indeed it is potentially even more useful – to develop equally sophisticated mechanisms for securely sharing and accessing data resources within and across organizations and to integrate such data resources into grid systems. There are several projects [12-20] - past and present, academic and industry - that focus on the data side of grid systems, developing what we call data grids.

Data grids take a number of different forms, and address data management issues such as transport, storage, wire integrity, security, replication, caching, and consistency. There are copy-in/copy-out systems that use FTP-like [21] mechanisms to make copies of the data at different locations, library-based approaches that provide special APIs to access remote data [13, 22, 23], complete distributed file systems [24], and interposition agents that mimic standard operating system behavior and redirect selected I/O operations to remote sources [9].

Each of the styles imposes different constraints, performance trade-offs, and programming burden on users. The most commonly used style to date has been copy-in/copy-out approaches as exemplified by the use of GridFTP [19]. The basic idea is simple. Before program execution, required data sets (including possibly the executable) are first copied from a remote location to a local file system. The application is executed, and specified output files are either copied to a remote location or are kept for local consumption.

There are several shortcomings to FTP-like approaches. Among them are redundant file copying, the need for accounts at multiple locations (if non-anonymous access is to be provided), and the cognitive burden placed on programmers to explicitly manage data transport, caching, and consistency issues. (A more detailed analysis is provided in Section 5.)

An alternative to FTP-like approaches is a Transparent Grid Data Management System (TGDMS) that emulates standard file system behavior while managing security, transport, caching, and consistency on behalf of the user. In a TGDMS the data resources in the grid are mapped into a directory-based namespace, which in turn is mapped into local operating system file system name spaces as a "mounted" directory. The result is that user applications can access data located throughout the grid without modifying their applications or scripts at all. Both programmers and end-users are completely isolated from even knowing the grid is there.

A common critique of TGDMSs is that the performance penalty for transparency is too high – and that it is not worth the cost. While "too high" is a subjective issue, the costs and performance penalty can be described quantitatively. In this paper we examine the performance of a commercially-available TGDMS, the Avaki Data Grid version 5.1[1], using a set of synthetic benchmarks that were designed to mimic four typical data access scenarios: a single client reading a local file, a single client reading a remote file, a single client writing a local file, and multiple parallel clients accessing

[1] Newer versions are now available – we believe that the main conclusions of this paper remain true.

a local file. The results were compared to the use of a Linux native NFS implementation. The outcome was that transparent read access can be achieved with either a small penalty (<10%) or – in the case of parallel access – a performance benefit. The story with write performance was not as promising - the penalty for writes was quite significant, typically a factor of eight slower. However, we believe that much of that can be attributed to implementation issues, and that more acceptable performance is possible.

The remainder of this paper is organized as follows: in Section 2 we present a brief description of the Avaki Data Grid architecture while the experimental design is discussed in Section 3. Performance results are detailed in Section 4. Section 5 presents related work and alternatives to TGDMSs.

2 The Avaki Data Grid

The Avaki Data Grid (ADG) [11-14][5, 14, 25, 26] is commercial grid software designed to simplify provisioning, access, and integration of data from multiple, heterogeneous, distributed sources. The primary goals of the Avaki Data Grid are 1) to deliver data to applications and users from a collection of geographically-distributed sources, and 2) to provide a single transparent interface to the data, facilitating data access, integration, and application development. Conceptually, the ADG architecture consists of three layers - an access layer, a transformation layer, and a provisioning layer.

The access layer provides a means for users and applications to access data in a location transparent fashion. This layer presents a familiar hierarchical namespace, where the interior nodes are directories, and the leaves are data objects – specifically files in our discussion here. Read/write access to files is provided via Avaki-specific NFS and CIFS servers, as well as via web interfaces, i.e., a browser. The NFS and CIFS servers are implemented by Avaki Data Grid Access Servers or DGAS. A DGAS presents the illusion of a single file system to client machines. By mounting the Avaki NFS or CIFS server, a client operating system essentially maps the Avaki namespace into the local operating system namespace, providing transparent access to remote resources. The DGASs are responsible for enforcing access control, caching data, and providing cache consistency.

The transformation layer supports the creation of abstract data objects, files, DBs, etc., that are the result of performing a transformation on other data objects. For example, filtering a file, merging and sorting two files, or performing an XSLT transformation. These transformations can be chained, and all of the resulting and intermediate data objects may exist in the namespace, have access control, and other metadata.

The provisioning layer maps local data sources, for example, file system directory trees, database tables and views, into the Avaki namespace. After checking permissions, the provisioning layer services data requests from the access and transformation layers.

Avaki has a grid-of-grids architecture. Each organizational unit, department, school, division, or company has one or more Avaki Grid domains. Grid domains are typically synonymous with administrative domains, i.e., authentication and identity

domains. Avaki grid domains may be joined together by connecting their namespaces and exchanging certificates. In the case that two domains are separated by a firewall, a proxy server can be set up, which routes the requests and responses between the provider and the consumer.

An ADG domain consists of four components: a grid domain controller (GDC), which manages the global namespace; one or more grid servers (GS), which handle transforms, meta-data, and the data catalog; share servers (SS), which export local files and directories into the ADG in the form of Avaki shares; and data grid access servers (DGAS), which provide access to the data in the grid, as well as data and meta-data caching. An example of an ADG domain is given in Fig. 1.

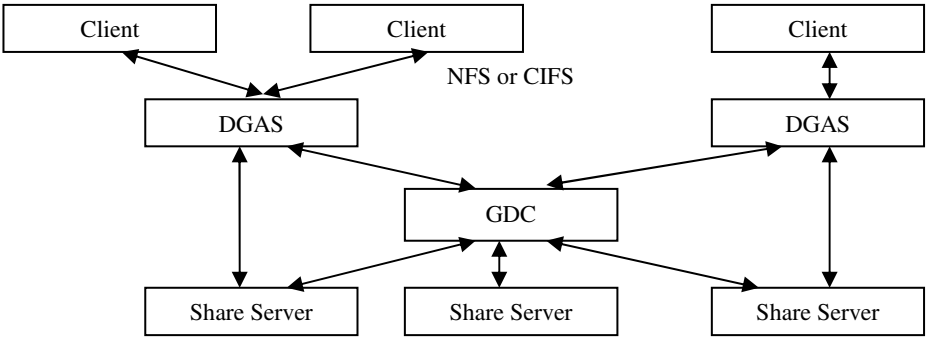


Fig. 1. Avaki Data Grid Architecture. A typical ADG domain consists of share servers that provision the data, a grid domain controller and zero or more grid servers that implement the name space and perform transformations, and one or more data grid access servers that provide cached access to client machines via NFS and CIFS.

Tracing a single read through the stack is illustrative of how the pieces work together. Suppose a client application issues a read on a file. The read is routed to the local operating system NFS or CIFS client. Assuming that the data are not in the local cache, the request is forwarded to a DGAS which first checks permissions. Access control information is a part of the file metadata. If the file metadata are not in the local DGAS cache, then the DGAS interrogates the GDC using the users' credentials, asking for the metadata and for a signed access cookie[2]. The GDC looks up the location of the data object, loads metadata from a database, and checks the credentials against the access control list. Once the DGAS has the metadata and signed cookie, it interacts directly with the share server for all subsequent operations. The DGAS then reads the data from the share server and returns the results to the client operating system, which in turn returns the data to the application. Note that the above steps are not necessary if the data are cached.

In order to expedite the data access, the DGAS has both an in-memory cache and disk cache. On reads the DGAS checks to see if the data are already in one of its

[2] The details of mapping from local identity spaces, e.g., Unix UID, to global name spaces, acquiring credentials, etc., is beyond the scope of this paper. Here we are giving a simplified overview that focuses on the data movement, not on security.

caches and still within the *coherence window*. If so, the cached data are returned. The coherence window is a property of each data object and is specified in seconds. If the data in the cache have been there longer than the coherence window, the DGAS must check with the share server to see if the data have been modified (a last write timestamp is part of the file metadata). If the data have been modified, the current cached data are evicted, and new data are retrieved and placed in the cache. If the data have not been modified, the window is reset and the read satisfied from the cache.

On writes, the DGAS cache use write-through; that is, the data are immediately written to the destination. The DGAS caches files in blocks. The block size, along with other DGAS parameters, can be dynamically tuned at run-time. The DGAS evicts cache blocks based on an LRU policy when the cache is full.

3 Experiment Design

3.1 Objectives

Our experiment was designed with three objectives. First, the experiment should reveal the file I/O performance of the Avaki Data Grid, namely, bandwidth and latency. Bandwidth was defined as the bytes read or written by one or more clients over a time period. Latency was the time from when a request was submitted until the response was available.

Second, only the performance of the ADG should be measured; that is, the experiment should exclude caching effects from the local file system. Towards this end, we intentionally flushed the cache by disconnecting the NFS mount point after the previous I/O operation and reconnecting the mount point before the next I/O operation. Without the calls of *umount* and *mount* to flush caches, the clients would see the results nearly instantaneously from the local file system caches.

Third, we wanted to measure the performance of the ADG under stress, i.e., how well the ADG responded when many clients sent requests at the same time. We did this using an MPI application described later. In this case, the aggregate bandwidth was defined as the bytes received by all clients before the slowest client finished its request. Thus, it was possible that some clients finished their tasks early and waited for the completion of the slowest client. We computed the aggregate bandwidth as the product of the number of clients and the bandwidth of the slowest reader, since each reader read the same file. This penalized ADG, but for a parallel application, it is more realistic since most parallel applications must wait for the slowest task.

The experiments were characterized from the perspective of the client. During the read experiments, the client read data from the DGAS via the local OS and the NFS protocol and simply discarded the transferred data. On writes, the client transferred data from its memory to the DGAS via the local OS and the NFS protocol. Both read and write operations were sequential whole file operations, i.e., the client accessed a file from beginning to end. Given the elapsed time and the file size, we could compute the latency and the bandwidth for the client.

The read/write operation was repeated immediately. The first operation was intended to measure the performance when the DGAS cache was cold while the second operation measured performance when the DGAS cache was warm. Note that the local OS caches were emptied between runs as mentioned above.

3.2 Experiment Implementation

There were several factors that could affect performance. We focused on the three that we observed had the largest impact - remote versus local data, file size, and the number of concurrent readers.

For our tests we generated files containing synthetic data. The files varied in size from 1 MB to 1 GB by powers of 2, i.e., 1, 2, 4, 8 MBs. We placed those files in share servers both locally and remotely.

To measure bandwidth we created a synthetic benchmark to read/write data from/to a file. Two versions of the benchmark were created – a sequential version and an MPI parallel version. Unbuffered I/O functions were used in order to eliminate the effects from buffered I/O utilities. Bandwidth was measured by dividing the bytes read or written by the elapsed time. Pseudo code for the body of the sequential benchmark is as follows:

```
long start_times[2], end_times[2];
get_timing(start_times);                          // start the timing
while (result = read(fd,buf,buffersize) > 0); // read the file
get_timing(end_times);                            // stop the timing
```

For any two consecutive reads/writes the code was repeated. Between two requests the local caches were flushed by unmounting and mounting the NFS mount point.

The second benchmark measured multiple readers within a cluster. We slightly modified the sequential benchmark to add an MPI barrier before and after each read operation, which released all clients at the same time. To compute bandwidth we multiplied the number of MPI tasks (readers) by the file size, and divided by the elapsed time. Pseudo code for the parallel benchmark follows:

```
long start_times[2], end_times[2];
MPI_Barrier(MPI_COMM_WORLD);      // wait for everyone is ready
get_timing(start_times);                          // start the timing
while (result = read(fd,buf,buffersize) > 0); // read the file
MPI_Barrier(MPI_COMM_WORLD);      // wait for everyone completes
get_timing(end_times);                            // stop the timing
```

C was used for the implementation. The code was compiled by gcc with default settings.

4 Performance

The performance evaluation environment consisted of two ADG domains. Avaki Data Grid release 5.1 was used. One domain was *TACCDataGrid*, at the Texas Advanced Computing Center (TACC) of the University of Texas at Austin, whose grid server was hosted by a dual-processor 3.06 GHz Intel Xeon CPU with 2 GB memory, running version 2.4.20-31.9smp of the Linux kernel. The other domain was *UVaCenturionGrid*, at the University of Virginia, which had a dual-processor 2 GHz AMD Opteron CPU with 2 GB memory, called *centurion-home*, and a cluster of 20 machines called *sunfire1*, *sunfire2*, etc. Each *sunfire* was a dual-processor 2.8 GHz Intel Xeon with 1GB memory. All machines at the University of Virginia were

running version 2.4.20-8smp of the Linux kernel. The two sites were connected via the Internet. Wide area throughput was limited by an OC-3 connection from the University of Virginia to the Internet and a 100 Mb connection from *UVaCenturionGrid* to the University of Virginia campus backbone.

We present and analyze the performance of the Avaki Data Grid under five scenarios. When we use the phrase "native NFS" in this paper, we mean the NFS V3 protocol with a native Linux NFS server. Avaki numbers are given using a Linux NFS client and an Avaki DGAS speaking NFS. For each scenario we ran the test 10 times before we computed the means and 95% confidence intervals.

Scenario 1: File open in LAN. The *centurion-home* hosted the GDC, a share server, and the DGAS. The reader, *sunfire1*, mounted the DGAS via NFS protocol just like any NFS client. In this scenario the DGAS handled requests as an NFS server. They were connected by Gigabit Ethernet. The latency for file open was measured.

Scenario 2: File read in LAN. The computers were connected as in Scenario 1. In this case, the bandwidth for a file read was recorded instead of latency. We read each file twice to reveal the effect of the cache in terms of bandwidth, and recorded both values.

Scenario 3: Simultaneous file read in LAN. The *centurion-home* hosted the grid server and share server. The difference was that more than one client read the file simultaneously. To be precise, 16 *sunfires* acted as concurrent clients. Each client had its own DGAS mounted locally via NFS protocol, i.e., each host had a separate DGAS that the host mounted and each DGAS communicated with the share server. All connections were Gigabit Ethernet.

Scenario 4: File write in LAN. Same as in Scenario 2, except that *sunfire1* became a writer. Because DGAS uses a write-through cache strategy, the write operations were performed only once.

Scenario 5: File read in WAN. Each domain had its own GDC. The source share server was hosted at TACC, as were the files to be read. The client, *sunfire1*, mounted to the DGAS on *centurion-home* via NFS. On read, *sunfire1* sent the read request to *centurion-home*, which forwarded the request to the grid server at TACC. In response, the share server from TACC sent the data back to *sunfire1* provided the user had permission to the data. We do not show in the paper the performance of file writes in WAN because the performance as expected suffered by the limited bandwidth and long latency of wide area network.

4.1 File Open in LAN

Latency was measured as the time for a client to establish a connection and open a file on a server. The client opened an existing file for reads. In case of writes, the client created a new file. Both the client and the server were in the local area network. Table 1 lists the file open latencies for native NFS and ADG in LAN. A file open for read in ADG needed three times as much time as native NFS, while a file open for write in ADG imposed even more significant overhead compared to NFS. We speculate that the reason is that ADG has to perform many steps with regard to metadata at several locations, such as share server, data grid access server and grid server. In contrast, metadata in NFS is handled at one location, the NFS server. As a result, NFS consumed less time and was more efficient.

Table 1. File open latencies. We ran the tests for ten times.

Read/Write	Latency (milliseconds)	95% Confidence Level
Native NFS read	1.1112	0.0385
Native NFS write	2.2947	0.1202
ADG read	3.5117	1.4938
ADG write	28.6897	11.9093

4.2 File Read in LAN

Corresponding to Scenario 2, the result of file read by an NFS client across a LAN is shown in Fig. 2. We can see that the bandwidth of the second read achieved 60 MB/s, an increase as much as six times of 10 MB/s bandwidth of the first read. The performance of the subsequent read improved greatly because the files were cached in the DGAS after the first read. Such a bandwidth increase for the second read sustained till the file size exceeded 1 GB, which exceeded the cache size of the DGAS.

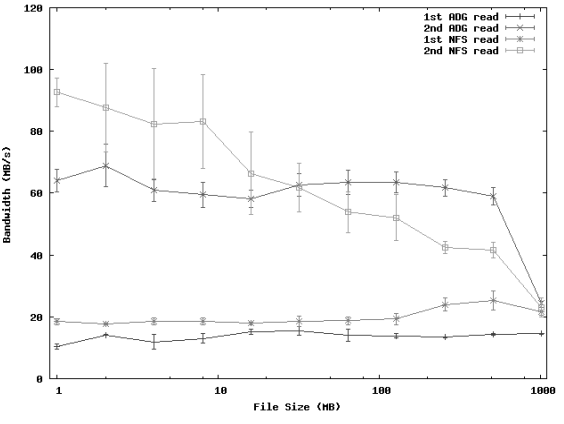


Fig. 2. File read in LAN for both ADG and native NFS. Average performance for each of the four tests is given, as well as the 95% confidence intervals.

With the intention of finding out the performance penalty that ADG pays for transparency, we always measured the performance of native NFS in the same setting. Fig. 2 also shows the performance of a native NFS client reading a file from an NFS server. For the first read, native NFS beat ADG by a margin of 5 MB/s. However, ADG outperformed native NFS by up to 20 MB/s for the second read, sustaining the bandwidth for files that were larger than 8 MB. This indicated that native NFS has a smaller cache than ADG, which makes use of disk space as file cache in addition to in-memory cache. Although writing data to the disk slowed down the first read, such aggressive caching of DGAS significantly benefited the second read on large files, and for files that were being read from a remote location. The performance differences in percentages are shown in Table 2 below. One can see that the native NFS usually beat the ADG by at least 15%, except on the second read of larger files.

Table 2. Difference in Percentage (%) = (1 - ADG/NFS) * 100%

File Size (MB)	Difference in Percentage (%)	
	1st Read	2nd Read
1	43.72495	30.94852
2	19.83195	21.25632
4	35.95397	26.16955
8	29.6479	28.51128
16	15.19068	12.40469
32	16.30578	-1.28616
64	24.50852	-17.8668
128	28.36182	-22.0486
256	44.22514	-45.4627
512	43.55048	-42.122
1024	33.19589	-6.48357

4.3 Simultaneous File Read in LAN

We chose a 128 MB file size for our simultaneous readers experiment. Both the performance of ADG and that of native NFS are shown in Fig. 3. It is well-known that native NFS does not scale when there are many concurrent reads. Our tests confirmed this fact: the aggregate bandwidth for NFS consistently stayed around 110 MB/s. So as more clients were added, each client got less bandwidth. When there were four clients, each one achieved a bandwidth of 27 MB/s but when there were 16 clients, each one only achieved about a bandwidth of 7 MB/s. In addition, there was no significant difference between the first and second read, which suggested that NFS cache has no effect for the second read because it is network bandwidth limited.

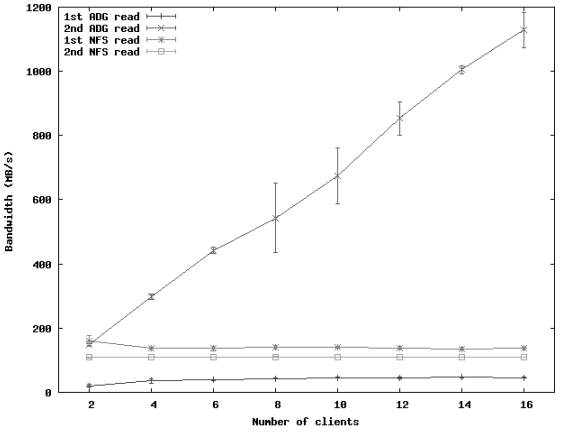


Fig. 3. Simultaneous file read. Averages are shown with 95% confidence intervals.

Although the first read for ADG was slower than that of native NFS, each ADG client could expect a much better performance at the second read, thanks to ADG caches. In ADG, the aggregate bandwidth of the first read maintained at a level of 45 MB/s for up to 16 clients, where the bandwidth was 2.8 MB/s for each client. However, for the second read, the aggregate bandwidth was linear with the number of clients. The more clients were added, the larger became the aggregate bandwidth. When there were four clients, the aggregate bandwidth was 298 MB/s. When there were 16 clients, a four times increase in the number of clients, the aggregate bandwidth was 1128 MB/s, an increase of 3.9 times in bandwidth. On average each client got a bandwidth of 70 MB/s, eight times what a native NFS client got when there were 16 clients. This suggested that ADG scales well when there is reuse, as for example, when bioinformatics codes are being repeatedly run against a large sequence or protein database.

4.4 File Write in LAN

For file write, both native NFS and ADG presented a fairly consistent performance for various file sizes shown in Fig. 4. In general, native NFS significantly outperformed ADG for write. A client could write a file at a bandwidth of 40 MB/s in native NFS, while at the bandwidth of 4 MB/s in ADG. ADG wrote data through to the share server without any caches. Thus, the performance suffered as more components were involved in ADG. We did not repeat file write twice as we did for file read because ADG directly writes data to the share server without any cache. Each time a client writes a file, he/she should expect the same bandwidth.

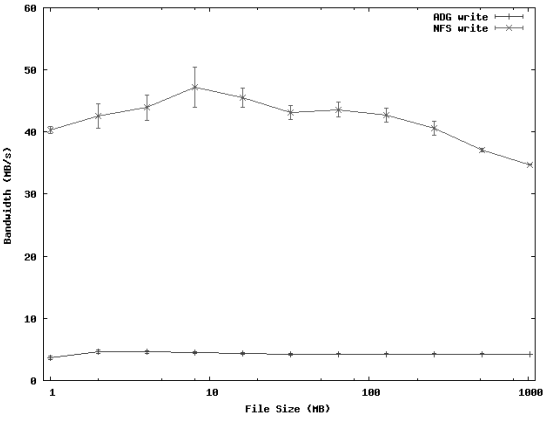


Fig. 4. File write in LAN. Averages are shown with 95% confidence intervals.

4.5 File Read in WAN

In Fig. 5 we present the bandwidth file read in the wide area network between the University of Virginia and TACC. As expected the first read was slow at a bandwidth close to 1 MB/s due to network limitations. The performance of the second read in

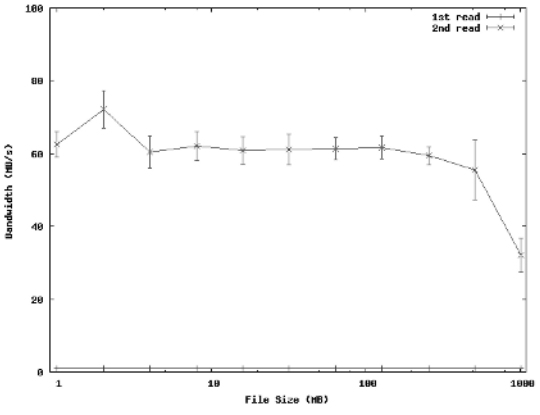


Fig. 5. File read in WAN. Averages are shown with 95% confidence intervals.

WAN was around 65 MB/s, which was comparable to that in LAN because the data were cached in the DGAS after the first read. This characteristic of ADG can greatly reduce the read latency and network load in cases that a lot of clients want to read the same set of files on the remote location. After the data are cached in the DGAS, the Avaki Data Grid is able to serve many clients with the local copy of the data without the need for retrieving one copy for each client. This saves significant network bandwidth resources. Most importantly, a client is able to access the data quickly and seamlessly regardless of the original locations. We do not compare ADG with native NFS here, because NFS V3 is not suitable for wide area networks, which will be explained in Section 5.

5 Related Work

The problems that data grids solve have been around for as long as networks have existed between computers. A number of solutions have been created to solve the problems of remote data access. In this section, we take a closer look at some of the more popular solutions and present their advantages and disadvantages. For each solution we will take up a case of a local user at one site trying to share a file with a remote user at another site. The first user, say Alice, is a user on the local machines owned by one company, whereas the second user, say Bob, is a user on the remote machines owned by another company. The two companies may not share a mutually-trustful relationship although the sharing of Alice's file with Bob has been approved.

5.1 Network File System – NFS

NFS V3 is the standard Unix solution for accessing files on remote machines within a LAN. With NFS, a disk on a remote machine can be made part of the local machine's file system. Accessing data from the remote system now becomes a matter of accessing a particular part of the file system in the usual manner. In our use-case above, Alice could run an NFS server on her machine, and Bob could run an NFS

client to mount Alice's file system onto his. Bob can now access the exact file that Alice wishes to share.

There are several advantages to NFS, the most significant of which is that it is easy to understand. Typically, Unix system administrators configure the server and client, and ordinary users like Alice and Bob simply use it without necessarily realizing that they are doing so. Moreover, applications need not be changed to access files on an NFS mount – the NFS server supports standard OS file system calls. Accordingly, files may be accessed entirely on in parts as desired. Finally, the NFS server and client tools come standard on all Unix's. On Windows, a special service pack must be purchased and installed.

The biggest disadvantage with NFS V3 is that it is a LAN protocol – it simply does not scale to WAN environments. If Alice and Bob are separated by more than a few buildings, using NFS between them becomes problematic. Moreover, if Alice and Bob belong to different organizations, as they are in our use-case, NFS cannot be deployed with reasonable guarantees of security.

Three characteristics of NFS doom it for use in wide-area, multi-organizational settings. First, the caching strategy on the NFS server typically releases data after 30 seconds and reloads the data on subsequent access. The result is a frequent retransmission of data and over-consumption of bandwidth. A related problem is that the read block size is too small, typically 8 KB. In a wide-area environment, latency can be high, therefore larger block sizes are needed to amortize the cost of the remote procedure call (RPC). Although the block size can be changed, most NFS V3 clients do not change it.

Second, and most seriously, NFS V3 does not address security well. An NFS V3 request packet is sent in the clear and contains the (integer) User ID (UID) and Group ID (GID) of the user making the read or write request. The NFS V3 server "trusts" the NFS client to not lie about the identity of the user making the request. Such a trustful relationship does not exist among multiple organizations, such as Alice's and Bob's. Even if the organizations do trust each other, man-in-the middle, imposter and snooping attacks can be made with NFS traffic. A VPN deployed between the organizations may attenuate some of these attacks, but VPNs introduce their own problems of management, trust and scalability. Further, firewalls typically do not permit NFS traffic through them making NFS impossible for cross-site use across firewalls.

Third, even assuming that the packets can be sent in a safe and trustworthy fashion, NFS requires that the identity spaces at the two sites be the same. In other words, not only should Alice and Bob have accounts on each other's machines, but Alice's UID on Bob's machine must be the same as her UID on her own machine. Likewise, Bob must synchronize his UIDs on his machine and Alice's machine in the same manner. Such synchronization would be possible if Alice and Bob were within a single domain; in our realistic use-case, they are not.

Other disadvantages plague NFS; we will mention them briefly here. NFS performance does not scale in a wide-area setting because it is a request-reply protocol which requires acknowledgments to be sent for every request, thus increasing effective transmission latency. NFS is a stateless protocol, i.e., the server does not keep track of the position of files being read. Accordingly, the server cannot pre-cache data or pre-position accesses to give clients better performance. Increasing

the number of clients overwhelms the one server deployed to serve data, thus reducing performance. In our use-case Alice wants to give Bob access to one of her files. If Bob also had some files he wished to share with Alice, he would have to run an NFS server on his machine and ask Alice to run an NFS client on hers. This kind of configuration can lead to a morass of cross-mounting, which can over-burden most administrators. In general, NFS requires $m \times n$ connections if m clients access data on n servers.

5.2 FTP and GridFTP

FTP has been the tool of choice for transferring files between computers since the 1970s. FTP is a command-line tool that provides its own command prompt and has its own set of commands. FTP may be used within a script; however, in that case, the password for the remote machine must be stored in a clear-text file on the local machine. Using ftp, Alice may connect to Bob, enter a username and password relevant to Bob's machine, change to the appropriate remote directory and then transfer the file.

The benefit of using ftp is that it is relatively easy to use, has been around for a long time and is therefore likely to be installed virtually everywhere. However, the disadvantages of ftp are numerous. First, Alice must have access to an account on Bob's machine, complete with username and password. Having such access means that Alice potentially could do more than just file transfer – she might be able to log in to Bob's machine and access files, directories and other machines to which she has not been given explicit access. From Alice's perspective, every transfer requires her typing the appropriate machine name, username and password. She could ameliorate some of this burden by using a configuration file for ftp, but that file may require storing a clear-text password for Bob's machine.

In order to eliminate some of these problems, Bob's site may choose to implement anonymous ftp. In this case, Alice need not have a username and password for Bob's machine, but must still remember the machine name and part of the directory structure. The problem with anonymous ftp is obvious – *anyone* may now access Bob's ftp directory, not just Alice. The potential for unauthorized overwriting or filling up of disk space is large.

FTP is also inherently insecure; passwords and data are transmitted in the clear. Snooping attacks may easily compromise Alice and Bob. Hence, most sites that have firewall protection shut down the standard ftp port to discourage such attacks. Even without firewalls, there are other disadvantages to using ftp. Because ftp requires making a copy of the data on Bob's machine, if Alice ever changes her own copy of the file, she must remember to ftp the new version of the file. Moreover, if Bob ever changes the file, he must remember to ftp the file back to Alice and reconcile concurrent changes, if any. This process is fraught with potential for inconsistencies, and the problem is compounded if additional people need to use Alice's file and receive versions from her at different points in time. Also, ftp is an all-or-nothing protocol – if even one bit of a large file changes, the entire file must be copied over. Finally, ftp is not conducive to programmatic access. Applications cannot use ftp to take advantage of remote files without significant modification.

SCP/SFTP belong to the *ssh* family of tools. SCP is basically a secure version of the Unix rcp command that can copy files to and from remote sites, whereas sftp is a secure version of ftp. Both are command-line tools. The syntax for scp resembles the standard Unix cp command with a provision for naming a remote machine and a user on it. Likewise, the syntax and usage for sftp resembles ftp.

The benefits of using scp/sftp are that their usage is similar to existing tools. Moreover, password and data transfer are encrypted, and therefore secure. However, a disadvantage is that these tools must be installed specifically on the machines on which they will be used. Installations for Windows are hard to come by. Moreover, scp/sftp do not solve several of the problems with ftp. In our use-case, Alice must still have access to an account on Bob's machine and she must continue to remember the appropriate machine name, username and password. Alice could ameliorate some of this burden by using an authorized keys file which permits password-less access, but she must then store her private key safely on her local machine.

Sites protected by firewalls may permit scp/sftp traffic on the designated port because the traffic is encrypted. However, scp/sftp does not attempt to solve the consistency problems of proliferating multiple copies of the file. Like ftp or rcp, a change of even one bit requires the entire file to be copied over. Finally, these tools are not conducive to programmatic access. Applications cannot take advantage of remote files using scp/sftp without significant modification.

GridFTP is a tool for transferring files built on top of the Globus toolkit [27]. GridFTP is an example of a service that characterizes the Globus "sum of services" approach for a grid architecture. Alice and Bob, in our use-case, could use GridFTP to transfer files from one machine to another, similar to the way they would use ftp. Naturally, both parties must install the Globus toolkit in order to use this service.

GridFTP solves the privacy and integrity of the problems with ftp by encrypting passwords and data. Moreover, GridFTP provides for high-performance, concurrent file transfer by design. An API enables accessing files programmatically, although applications must be re-written to use new calls. Data can be accessed in a variety of ways - for example, blocked and striped. Part or all of a data file may be accessed, thus removing the all-or-nothing disadvantage with ftp.

However, GridFTP does not address the identity space problems with ftp. Alice and Bob in our use-case must still have an account on each other's machine, thus giving them more privileges than just file access. Instead of a machine name, username and password as in ftp, Alice and Bob have to remember just the machine name. Their identities are managed by Globus using session-based credentials. Finally, GridFTP does not solve the problems of maintaining consistency between multiple copies, because Alice and Bob would still be required to maintain at least two copies of the file, one on each user's machine.

5.3 NFS over IPSec

IPsec is a protocol devised by IETF to encrypt data on a network. With IPSec installed and configured properly, all traffic on a network can be encrypted. Consequently, illegitimate snooping of network traffic does not affect the privacy and integrity of the communication between a server and a client. NFS over IPSec implies traffic between an NFS server and an NFS client over a network on which

data has been encrypted using IPsec. The encryption is transparent to an end-user. NFS over IPSec removes some, but not all, of the disadvantages of using NFS.

NFS over IPSec results in encrypted NFS traffic, thus regaining privacy and integrity. However, NFS continues to be a LAN-based protocol which does not scale to the WAN-like environment typical in our use-case. All of the performance, scalability, configuration and identity space problems we discussed earlier remain. In addition, in order to deploy IPSec, all of the machines in Alice's and Bob's domains must be reconfigured. Specifically, their kernels must be recompiled in order to insert IPSec in the communication protocol stack. This recompilation is hard; anecdotal evidence suggests that the recompilation is risky, error-prone and ill-documented. Finally, once this recompilation is done, *all* traffic between all machines is encrypted. Even web, email and ftp traffic is encrypted whether desired or not.

5.4 De-Militarized Zone – DMZ

A DMZ is simply a third set of machines accessible to both Alice and Bob using ftp or scp/sftp, established to create an environment trusted by both parties. When Alice wishes to share a file with Bob, she must transfer the file to a machine in the DMZ, inform Bob about the transfer and request Bob to transfer the file from the DMZ machine to his own machine. Although both Alice and Bob have relatively unfettered access to the DMZ machines, neither party compromises his/her own machines by letting the other have access to them.

With a DMZ, neither Alice nor Bob requires an account on the other's machines. Typically, companies deploying DMZs also deploy scp/sftp or some such secure means of file transfer. Therefore, these tools must be installed on all concerned machines. Alice and Bob both have to remember machine names, usernames and passwords for the DMZ machines. However, they now have to remember an additional step of informing the other whenever a transfer occurs.

DMZs worsen the consistency problems by maintaining three copies of the file. Also, because the file essentially makes two hops to get to its final destination, network usage increases. DMZs may address security concerns, but they do not ameliorate any of the other problems with scp/sftp and they do increase administrative burden. If Alice's company decides to co-operate with a third company, thus requiring Alice to interact with Chris at that company, she must now create and remember yet another DMZ configuration for interacting with Chris. The Alice-Bob DMZ cannot be reused because of the potential for Chris to access files intended for Bob.

5.5 Andrew File System – AFS

The Andrew File System is a distributed network file system that enables access to files and directories distributed across multiple sites. Access to files involves becoming part of a single virtual file system. AFS comprises several cells, with each cell representing an independently-administered file system. In our use-case, the file system on Alice's machine would be one cell, whereas the file system on Bob's machine would be another. The cells together form a single large virtual file system that can be accessed similar to a Unix file system.

AFS permits different cells to be managed by different organizations thus managing trust. In our use-case, Alice and Bob would not require accounts on the other's machines. Also, they could control each other's access to their cell using the fine-grained permissions provided by AFS. When Bob accesses one of Alice's files for which he has permission, he accesses exactly the current copy of the file. Thus, AFS avoids the consistency problems with other approaches using copy-on-open semantics unless there are multiple concurrent writers (which AFS does not deal with well.) In order to improve performance, AFS supports intelligent caching mechanisms. Since access to an AFS file system is almost identical to accessing a Unix file system, users have to learn few new commands, and legacy applications can run almost unchanged.

AFS implements strong security features. All data are encrypted in transit. Authentication is using Kerberos, and access control lists are supported.

The drawbacks of AFS revolve around the use of Kerberos and the fact that it is a file system. Let me explain. The use of Kerberos means that *all* sites and organizations that want to connect using AFS must themselves use Kerberos authentication *and* all of the Kerberos realms must trust each other. In practice this means changing the authentication mechanism in use at the organization. This is a non-trivial and typically politically very difficult step to accomplish. Second, the realms must trust each other. This is similarly difficult to accomplish. Third, the Kerberos security credentials time-out eventually. Therefore, long-running applications must be changed to renew credentials using Kerberos's API. Also, AFS requires that all parties migrate to the same file system. In other words, Alice and Bob would have to migrate their entire file systems to AFS, which would probably be a significant burden on them and their organizations.

6 Conclusion

The question we asked at the beginning of the paper is whether a grid file system can achieve transparency without significant compromise in performance. To answer it, we measured the performance of file I/O in the Avaki Data Grid and compared it to "native" NFS performance. The results were mixed, though encouraging. For single client local file operations, native NFS outperformed the ADG by 15% to 45% for smaller files, though for files larger than 32 MB ADG outperformed native NFS. For writes ADG was significantly slower than the native NFS – it was not quite clear why. On the other hand, for concurrent readers ADG outperformed native NFS by as much as a factor of five.

The big win is in remote data access. Avaki's cache makes subsequent access significantly faster than re-transmitting the data. While end-users could explicitly manage their own cache, in our experience they don't. The result is either significant resending of data that has not changed, or users accessing out of date data inadvertently[3].

[3] The authors have seen an internal report of a top 10 pharmaceutical company that reports that 40% of all jobs end up being recomputed because the wrong – or more often old – data were used. Thus the jobs need to be re-executed, consuming resources, and delaying drug development.

In conclusion, although the data grid we tested did introduce some overheads, it does not seem, in our opinion, an unreasonable price to pay in order to achieve transparency and scalability. By quantifying I/O performance in this paper, we hope to help researchers and IT professionals gain an insight into the trade-offs between transparency and performance in data grids.

Acknowledgments

We would like to thank several people for their help, without which those experiments could not have taken place: Scott Ruffner and Mark Morgan at the University of Virginia, Jay Boisseau at the Texas Advanced Computing Center of the University of Texas at Austin, Jerry Perez at Texas Tech University, and Chuck Kesler at North Carolina BioGrid.

References

1. Berman, F., G.C. Fox, and A.J.G. Hey, Grid Computing: Making the Global Infrastructure a Reality. Wiley Series in Communication Networking & Distributed Systems. 2003: Jon Wiley & Sons.
2. Abbas, A., Grid Computing: A Practical Guide to Technology and Applications. 2004: Charles River Media.
3. Foster, I. and C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure. 1999: Morgan Kaufman Publishers.
4. Foster, I., et al., The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. 2002.
5. Grimshaw, A.S., et al., From Legion to Avaki: The Persistence of Vision, in Grid Computing: Making the Global Infrastructure a Reality, Fran Berman, Geoffrey Fox, and T. Hey, Editors. 2003.
6. Grimshaw, A. and A. Natrajan, Legion: Lessons Learned Building a Grid Operating System. Transactions of the IEEE, 2005.
7. Lewis, M.J., et al., Support for Extensibility and Site Autonomy in the Legion Grid System Object Model. Journal of Parallel and Distributed Computing, 2003. Volume 63: p. pp. 525-38.
8. Frey, J., et al., Condor-G: A Computation Management Agent for Multi-Institutional Grids. Journal of Cluster Computing, 2002. 5: p. 237-246.
9. Thain, D., T. Tannenbaum, and M. Livny, Condor and the Grid, in Grid Computing: Making The Global Infrastructure a Reality, F. Berman, A.J.G. Hey, and G. Fox, Editors. 2003, John Wiley.
10. Foster, I. and C. Kesselman, Computational Grids, in The Grid: Blueprint for a New Computing Infrastructure. 1999, Morgan-Kaufman.
11. Grimshaw, A.S., The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM, 1997. 40(1): p. 39-45.
12. Kola, G., et al., DISC: A System for Distributed Data Intensive Scientific Computing. in Proceedings of the First Workshop on Real, Large Distributed Systems (WORLDS'04). 2004. San Francisco, CA.
13. White, B., A. Grimshaw, and A. Nguyen-Tuong, Grid Based File Access: The Legion I/ O Model. in Proceedings of the Symposium on High Performance Distributed Computing (HPDC-9). 2000. Pittsburgh, PA.

14. Grimshaw, A., Avaki Data Grid - Secure Transparent Access to Data, in Grid Computing: A Practical Guide To Technology And Applications, A. Abbas, Editor. 2003, Charles River Media.
15. OGSA-DAI project home page. [cited; Available from: http://www.ogsadai.org.uk/.
16. White, B., et al., LegionFS: A Secure and Scalable File System Supporting Cross-Domain High-Performance Applications. in Proceedings SC 01. 2001. Denver, CO.
17. DAIS Working Group. [cited; Available from: https://forge.gridforum.org/projects/dais-wg.
18. OGSA Byte I/O Working Group. [cited; Available from: https://forge.gridforum.org/projects/byteio-wg.
19. 19.Allcock, W., GridFTP Protocol Specification (Global Grid Forum Recommendation GFD.20). 2003 [cited; Available from: http://www.globus.org/alliance/publications/papers.php#GridftpSpec02.
20. Bester, J., et al., GASS: A Data Movement and Access Service for Wide Area Computing Systems. in Sixth Workshop on I/O in Parallel and Distributed Systems. 1999.
21. Postel, J. and J. Reynolds., RFC 959 - File Transfer Protocol. 1985.
22. Rajasekar, A., et al., Storage Resource Broker - Managing Distributed Data in a Grid. Computer Society of India Journal, Special Issue on SAN, 2003. 33(4): p. 42-54.
23. GGF, Simple API for Grid Applications (SAGA), Global Grid Forum.
24. Satyanarayanan, M., Scalable, Secure, and Highly Available Distributed File Access. Computer, May 1990. 23: p. 9-18,20-21.
25. Avaki, http://www.avaki.com/.
26. Avaki, Avaki Data Grid Administration Guide, Release 5.1. May 2004.
27. Globus, http://www.globus.org/.

A Topology Self-adaptation Mechanism for Efficient Resource Location*

Luis Rodero-Merino[1], Luis López[1], Antonio Fernández[1], and Vicent Cholvi[2]

[1] LADyR, Universidad Rey Juan Carlos, 28933, Móstoles, Spain
{lrodero, llopez, anto}@gsyc.escet.urjc.es
[2] Universitat Jaume I, 12071, Castellón, Spain
vcholvi@lsi.uji.es

Abstract. This paper introduces a novel unstructured P2P system able to adapt its overlay network topology to the load conditions. The adaptation is performed by means of a mechanism which is run by the nodes in the network in an autonomous manner using only local information, so no global coordinator is needed. The aim of this adaptation is to build an efficient topology for the resource discovery mechanism performed via *random walks*. We present the basis of the adaptation mechanism, along with some simulation results obtained under different conditions. These results show that this system is efficient and robust, even in front of directed attacks.

1 Introduction

The Peer-to-Peer (P2P) paradigm has brought new communication opportunities for Internet users. P2P systems present advantages like flexibility, scalability and fault tolerance, thanks to the lack of central coordinators or controllers. But this same lack of central entities has brought new technical challenges.

Maybe one of the key issues to be solved is how to locate resources efficiently. Several solutions have been proposed, each with its advantages and drawbacks. It seems that P2P systems with *unstructured overlay networks* are suitable for certain scenarios like mass-market distributed resource sharing.

Unfortunately, it is not trivial to offer efficient search solutions in unstructured networks. Flooding based proposals (like the first versions of Gnutella) present scalability issues. Because of this, the research community is making efforts to develop solutions based on *random walks*. More specifically, to combine random walks with *dynamic overlay topologies* (topologies that change during the system life) is an approach that has lead to promising results [1, 2].

Here we introduce a solution based on the same idea of using random walks along with a dynamic topology. Changes on the topology are performed by the nodes themselves to adapt it to the load on the network. In order to do this, each node runs a

* This work was partially supported by the Spanish Ministry of Science and Technology under Grant No. TSI2004-02940 and TIN2005-09198-C02-01, by Bancaixa under Grant No. P1-1B2003-37 and by the Comunidad de Madrid under Grant No. S-0505/TIC/0285. The authors would like to thank Gia creators Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham and Scott Shenker for providing us with their simulator source code.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 660–671, 2006.
© Springer-Verlag Berlin Heidelberg 2006

reconnection mechanism (the same for all nodes) that periodically computes to which other peers the node must connect. This solution has been implemented in a system called **DANTE**$_2$[1].

This paper is organized as follows. In Sect. 2 we revise some current solutions for resource location in P2P networks. Section 3 presents previous works which form the conceptual basis of DANTE$_2$. Section 4 describes the adaptation mechanism used by DANTE$_2$. Section 5 presents some results obtained by simulation. Those results measure the performance of DANTE$_2$ under a variety of circumstances. Finally, Sect. 6 contains the conclusions of this paper and possible lines of future work.

2 Resource Discovery in P2P Systems

Traditionally, solutions for resource location in P2P systems are classified in two groups: *centralized* and *decentralized*. In centralized solutions a central repository stores an index of all resources in the network (like in Napster [3]). This approach makes the system vulnerable to attacks or censorship and poses scalability issues.

In decentralized systems, on the other hand, the resource discovery service is provided by the peers themselves. Decentralized systems are usually classified by the kind of *search mechanism* they implement to route search messages through the network:

1. *Structured systems*. These systems use specialized placement algorithms to assign responsibility for each resource to specific peers, as "directed" search mechanisms to efficiently locate resources. One example is Chord [4].
2. *Unstructured systems*. These systems do not have precise control over the resource placement and, traditionally, use search mechanisms based on flooding, random walks or supernodes. Examples are Gnutella and KaZaA.

Structured systems are very efficient: they usually require few communication steps to find some resource, and do not produce false negatives (i.e., the search fails only if the demanded resource is not in the system). On the other hand, unstructured systems tend to be less efficient, and may yield false negatives.

Yet, unstructured systems have some advantages: they have little management overhead, adapt well to the transient activity of P2P nodes, take advantage of the spontaneous replication of popular content and allow to perform queries by keyword in a simpler way than with directed search protocols. These advantages seem to make unstructured systems suitable for many real-world situations, like massive file sharing systems. Further discussion comparing structured and unstructured systems can be found in [5].

2.1 Resource Location in Unstructured P2P Networks

Three search methods are typically used for resource location in unstructured networks. The first one is *flooding*, where each peer broadcasts the queries to all its neighbors. It is well known [6] that the flooding solution presents problems of scalability.

[1] From Dynamic self-Adapting Network TopologiEs.

Another solution is based on the idea of *superpeers*. These are special nodes that store the index of resources shared by the rest of peers. The eDonkey [7] network, for instance, uses this approach. Yet, these systems are dependant on the availability of those powerful enough nodes.

Finally, the third search mechanism is based on *random walks*. Here, nodes forward each query to only one peer, chosen randomly among its neighbors. There is little communication overhead compared with flooding, but it can take longer to solve queries. In [8] and [9] we can find some studies that state that random walks seem to be a promising technique suitable to solve the scalability problems of flooding.

2.2 Dynamic Topologies Based Proposals

It is well known that the performance of random walks is highly dependant on the topology of the overlay network [10, 11, 12]. Thus, some solutions have been proposed that try to adapt the overlay topology to the network load, in order to improve the efficiency of the search process.

First, Lv *et al.* [2] introduced a P2P system in which nodes avoid congestion by means of a flow control mechanism that redirects the most active connections to neighbors with spare capacity. Another work is **Gia** [1], proposed by Chawathe *et al*. In Gia, queries are forwarded to high capacity nodes. An active flow control mechanism avoids overloading hot spots: each node notifies its neighbors the number of queries they can send to it, which depends on its spare capacity. Topology is adapted by a mechanism based on nodes' *level of satisfaction*, which measures the distance between a node's capacity to the sum of its neighbors capacities, normalized by their degrees. This parameter determines whether or not each node will adapt the topology, and the frequency of these adaptations.

DANTE$_2$ implements a different reconnection mechanism that we deem can lead to even more efficient topologies. First of all, DANTE$_2$ is inspired on the results of Guimerà *et al.* [13] on the relationship between network topologies and search performance (see Sect. 3). Another difference is that nodes in DANTE$_2$ do not keep track of their neighbors' state, nor implement any explicit flow control technique. Thus DANTE$_2$ avoids the communication overhead due to those activities. Some simulations comparing DANTE$_2$ and Gia are presented in Sect. 5.

3 Previous Work

The self-adaptation mechanism used in DANTE$_2$ is inspired on the results of Guimerà *et al.* [13] and on the algorithm proposed by Cholvi *el al.* [10]. Guimerà *et al.* were able to characterize the topologies that, given a search mechanism based on random walks, minimize the average time needed to perform a search. They found that when the system is not congested, the optimal topology is a star-like structure. Furthermore, they also found that when the system is congested, the optimal topology is a random-like one. However, Guimerà *et al.* did not state how these topologies could be achieved dynamically in a real system. In P2P networks, we face two problems: the lack of global knowledge, and the absence of a coordinator that tells nodes to which other peers they must connect to.

Thus, applying Guimerà results to P2P networks is not straightforward. A topology adaptation mechanism that fits P2P systems should be run locally at the nodes, and should not need global knowledge. Cholvi *et al.*, in [10], proposed a first mechanism that, depending on the current system load, makes nodes to locally change their connections so that the obtained topologies are random for high loads and star-like for low loads. Yet, in that solution nodes need to now all other peers state.

Finally, both in [13] and in [10] it is assumed that all participants have the same capacities, that is, the network is *homogeneous*. Nonetheless, nodes in real networks are known to be *heterogeneous* [14].

A previous version of DANTE$_2$ was introduced in [15]. Although the core idea of that work was the same used here (using a self-adapting topology to improve searches efficiency), this paper introduces key improvements: a better, more accurate reconnection mechanism that takes into account the nodes heterogeneity, and more complete simulations in different and more realistic scenarios.

4 DANTE$_2$ Self-adaptation Mechanism

In DANTE$_2$ each peer knows its own resources as well as the resources held by its neighbors. Based on this, it is easy to understand that nodes are more interested on being connected to peers with many neighbors. Therefore, DANTE$_2$ encourages peers to establish connections with high degree nodes. DANTE$_2$, in fact, aims to form highly clustered (even centralized) topologies. However, this holds only as long as highly connected nodes can handle all the incoming traffic.

Taking into account this reasoning, DANTE$_2$ uses an algorithm that, when the network traffic is low, drives the network to a star-like overlay topology. Thus, searches can be answered in only one hop, since the central nodes know all the resources in the system. In turn, when the number of searches increases, well-connected nodes will become congested and their neighbors will start to disconnect from them. Hence, this will drive the network to a more random-like topology that, although it makes search messages to traverse longer paths to find some resource, will balance the load and perform better than by using a highly congested central node.

More specifically, in DANTE$_2$ each node can establish connections to other nodes. We say that a connection is *native* for the establishing node and *foreign* for the accepting node. Nodes can change their native connections, but not their foreign ones. Furthermore, each node periodically runs a reconnection mechanism with which native connections are changed. This mechanism firstly obtains a list of potential candidates C to which it can connect. Then, it assigns a probability p_i to each candidate i, and chooses candidates at random using their respective probabilities. Finally, the peer reconnects its native connections to the chosen candidates.

The probability assigned to a candidate $i \in C$ is based on its "attractiveness", denoted as Π_i and defined as

$$\Pi_i = k_i^{\gamma_i} \tag{1}$$

where k_i is the degree (number of neighbors) of peer i, and γ_i is computed as

$$\gamma_i = 2 \times c_{i_{norm}} \times (1 - t_{i_{norm}}) \tag{2}$$

$c_{i_{norm}}$ is the normalized processing capacity of node i, where the normalization is performed as follows. Let c_i be the capacity of node i, and $c_{max} = \max_{i \in C}\{c_i\}$. Then,

$$c_{i_{norm}} = \frac{c_i}{c_{max}} \tag{3}$$

it follows that $0 < c_{i_{norm}} \leq 1, \forall\, i$, where a larger $c_{i_{norm}}$ means that node i is more attractive as it has more capacity to process searches.

$t_{i_{norm}}$ represents the average time spent by a search at node i (time in queue plus processing time), normalized. The normalization is computed as follows. Let t_i be the mean search processing time of node i, $t_{max} = \max_{i \in C}\{t_i\}$ and $t_{min} = \min_{i \in C}\{t_i\}$. Then,

$$t_{i_{norm}} = \frac{t_i - t_{min}}{t_{max} - t_{min}} \tag{4}$$

It is straightforward to see that $0 \leq t_{i_{norm}} \leq 1\ \forall\, i$, where a lesser $t_{i_{norm}}$ means that node i is more attractive as it takes less time for searches to be served.

Finally, once the Π_i values are computed for all candidates in C, each candidate $i \in C$ is assigned a probability p_i of being chosen that is computed as

$$p_i = \frac{\Pi_i}{\sum_{j \in C} \Pi_j} \tag{5}$$

By the definition of Π_i (Eq. 1), the attractiveness of node i is strongly dependant on its degree k_i. The higher the degree, the more attractive the node becomes, and so more peers will try to connect to it, increasing again k_i (and therefore Π_i). This process leads quickly to centralized topologies. The form of γ_i, on the other hand, comes from the fact that the reconnection function must favor high capacity nodes (capacity is represented by c_i), and avoid loaded peers (load is given by t_i). Thanks to this reconnection mechanism, the system behaves in an adaptative manner, changing its topology to suit the load conditions.

Candidates Sampling. The reconnection mechanism of DANTE$_2$ depends on the set of candidates C to which the node can connect. There are several mechanisms that could be used to build this list of candidates. For example, a *gossiping* based service like [16] could spread information about nodes in the network. Another solution is to make nodes to keep a *cache* of other peers in the network.

DANTE$_2$ implements a third solution. Whenever a node starts a new reconnection, it launches a special *Look For Node* message, that traverses the network following a random walk with a bounded TTL. When the TTL expires, another message is sent to the source node with the list of traversed peers. This list becomes the set of candidates. This technique has small incidence on the network load, and Newman's results [17] show that the set obtained is a good sample of the overall network.

5 Simulations

To study DANTE$_2$'s performance we have developed a simulator that implements its reconnection mechanism. Simulations use the microsecond as the minimum unit of time.

The capacity of each node is set by two parameters: *bandwidth* and *processing capacity*. Nodes perform tasks, like the processing of an incoming message or an internally started process (e.g., the triggering of a new reconnection). When performing some task the node is said to be *busy*. Any other pending task in the node is enqueued until the present task is finished.

The processing time t_{proc} depends on the tasks being performed. Tasks other than searching for a resource in the lists of known resources are assumed to take one unit of time. Searches for resources take a time proportional to the number of resources checked m and the node's processing capacity c_i, $t_{proc} = \frac{m}{c_i}$. Some tasks need to send a message. The duration of sending a message , t_{send}, depends on the node's bandwidth b_i and the packet size s, $t_{send} = \frac{s}{b_i}$. Finally, the time the node is busy, t_{busy}, for one task is computed as $t_{busy} = \max\{t_{proc},\ t_{send}\}$. This time is not $t_{proc} + t_{send}$, because we assume that the sending of messages and the processing of searches run in a pipeline. Nodes capacities and bandwidths are assigned following the distribution depicted in Table 1. This distribution is derived from the measured bandwidth distributions of Gnutella nodes reported in [14].

Table 1. Capacities and upload bandwidths distribution for simulations

Capacity level	Percentage of nodes	Processing capacity c_i	Bandwidth b_i
1x	20 %	0.1	0.01
10x	45 %	1	0.1
100x	30 %	10	1
1000x	4.9 %	100	10
10000x	0.1 %	1000	100

Each node starts a new search for a random resource periodically. The *time between searches*, tbs, is a parameter of the simulation that allows to set the load on the system. Each node holds 100 resources. All resources have the same popularity (no resource is more likely to be looked for than other). The *replication rate* r is another simulation parameter, that states the rate of nodes that hold each resource (in percentage).

Nodes manage 10 *native* connections each. Reconnections are triggered every 30 seconds of virtual time. Nodes change 5 *native* connections at each reconnection. We assume there is an external service that provides peers, at start-up time, with a list of some other nodes present in the system. When some peer is started it chooses its initial neighbors randomly from the list provided by that service. Hence, all experiments start with a random topology. Similarly, if a *native* connection points to some node that leaves the network (is attacked or deactivated), that connection is redirected to another peer chosen at random from a list again obtained from the external service.

The *Look For Nodes* messages have a TTL of 30 hops. Resource search messages have a TTL of 1000 hops. Both values were chosen empirically. The first one proved to be enough to get a good sampling of the network, the second one allowed to obtain a high success rate, both for DANTE$_2$ and Gia simulations.

Topology Evolution in DANTE$_2$. First, we study how in DANTE$_2$ the system is able to adapt itself, changing its topology as the (virtual) time passes. The results of two simulations are shown, with two different replication rates: $r = 0.01$ and $r = 0.05$. Both simulations are run with 10000 nodes (so $r = 0.01$ implies that each resource is held by only one node) and a time between searches $tbs = 1$ second. Simulations were run for 60 minutes of virtual time. All searches finished successfully in both simulations.

In Fig. 1.(a) we see how the mean number of hops changed as the virtual time passed. In the X axis we represent the virtual time, in minutes. In the Y axis we represent the average number of hops that took to solve searches started during the corresponding minute of virtual time. The number of hops decreases readily as the time passed, until it is stabilized to 1 after some minutes (tens of reconnections). This means that the network has reached a centralized topology, starting from a random one, and all searches are solved in just one hop.

On the other hand, we see in Fig. 1.(b) how the topology evolution makes the average search time to decrease. DANTE$_2$ builds an efficient topology, where searches are completed in very little time (about 30 milliseconds).

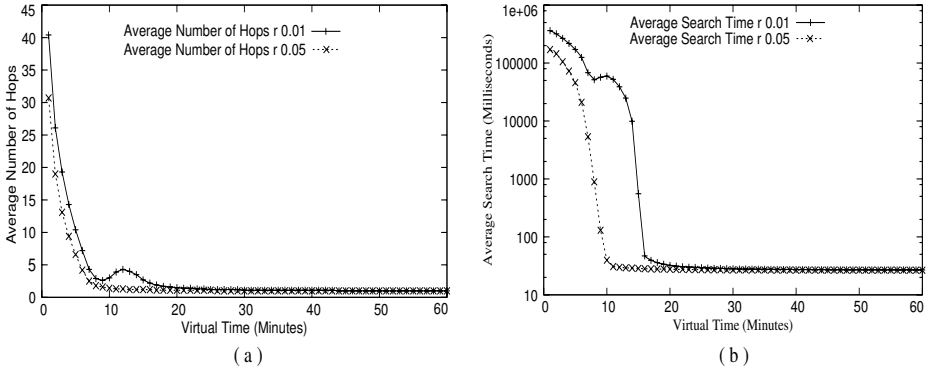


Fig. 1. Average number of hops and searches duration

Robustness Against Peers Churn. It is well known that peers may enter and leave the network at a high rate. This can, in some cases, compromise the efficiency of the system. In this section we present some simulations results that show how DANTE$_2$ performance is not strongly affected by the churn of peers.

The simulations of this section are run with 10000 nodes. The replication is $r = 0.05$ and the time between searches is $tbs = 5$ seconds. Only searches started between minutes 31 and 60 (included) are taken into account for the results. Searches started before minute 31 are discarded to avoid the initial transition state. The simulations were run until all searches started before minute 61 were finished.

Initially, nodes are active with a probability of 0.5. At start time, active nodes form a random topology. Each active node will run for a certain time that is independently calculated using a exponential distribution. When the time expires, the node is deactivated: it discards all searches in its queue, and closes its connections with all its neighbors.

After 0.5 seconds of virtual time, the node changes to the active state again, pointing its *native* connections to 10 nodes chosen at random, and the time to remain active is recalculated. This is similar to simulate nodes leaving the system as other nodes simultaneously joining it (a similar approach is used in [1]).

To simulate different churns, we set different values for the mean of the exponential distribution used to compute the time nodes will stay active: 1, 5, 10, 50 and 100 minutes. Finished searches are classified into three categories: *successful*, *failed* (the search TTL expired before the resource was found) or *discarded* (search was at a node that changed to the deactivated state). Figure 2 shows searches results for each experiment.

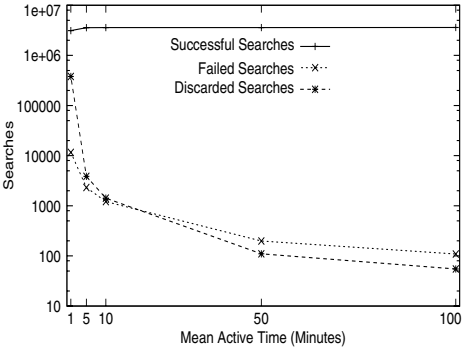


Fig. 2. Search results under churn

For the higher churn, the number of discarded plus failed searches is about 10% of the total. Yet, when the mean of the distribution increases, the number of discarded and failed searches decreases sharply. When the mean is 10 minutes, the number of unsuccessful searches (approximately 2600) represents only the 0.07% of the total.

In Figs. 3.(a) and 3.(b) it is shown how the churn of peers affects the search performance (only for successful searches). Although the number of hops and the time to complete queries increase, they are within what we deem are acceptable bounds even when the churn of peers is high.

Robustness Against Attacks and Congestion Avoidance. In DANTE$_2$, high capacity nodes tend to have more connections, even forming a starlike topology. Thus, it can be argued that DANTE$_2$ is vulnerable to attacks targeted to well-connected nodes, or that those nodes could become congested and so compromise the system performance. In this section we discuss how attacks can affect DANTE$_2$'s behavior, and how congestion is avoided by the reconnection mechanism. The simulations presented here were run with 10000 nodes, and replication $r = 0.01$. Two different loads were tested.

The results are shown in Figs. 4.(a) and 4.(b). Both of them show how the network behavior evolves as the virtual time passes, from the first minute of simulation (remember that initially nodes form a random topology) to minute 90. At minute 30, when the network has moved to a centralized topology, an attack is performed: the 10 best connected nodes (central nodes) are forced to leave the network. Those are also the

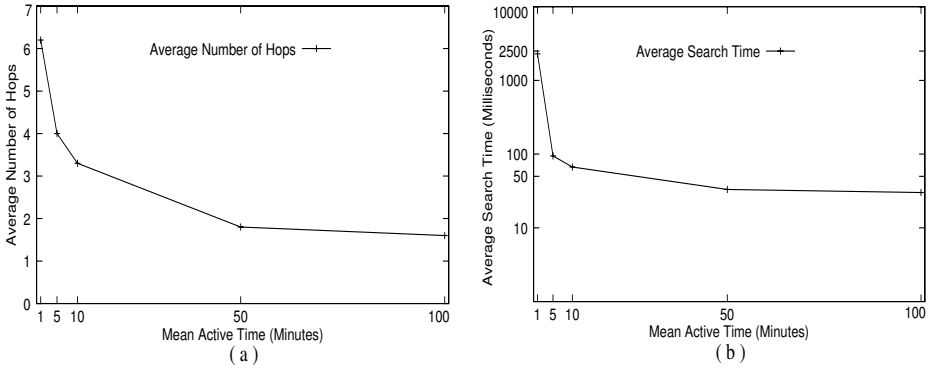


Fig. 3. Searches hops and duration under churn

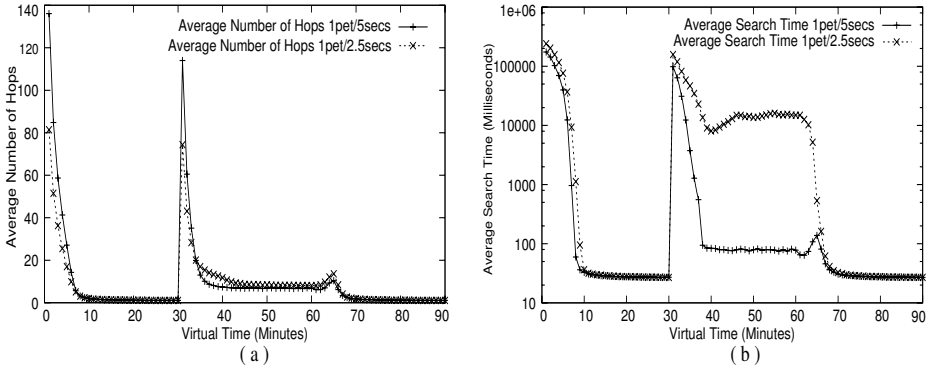


Fig. 4. Searches hops and duration under attack

10 most capable nodes in the network (see Table 1). 30 minutes later, those nodes are reactivated. We will check how DANTE$_2$ reacts to those events.

Figure 4.(a) shows how the average number of hops to find resources decreases sharply in a few reconnections until it reaches a value close to 1. At that moment the network has a centralized topology. When the attack is performed at minute 30, the remaining nodes redirect their connections randomly, so a random topology appears again. From that moment on, nodes will try to connect to the remaining peers with higher capacity (in this case, nodes of the fourth Capacity Level at Table 1). The network is never centralized again, as there are no nodes with enough capacity to become central. Here we can see how DANTE$_2$ actively avoids overloading nodes, not allowing them to receive too many connections if they can not handle them. Yet, highly connected nodes appear so the mean number of hops decreases sharply in a few minutes (to lesser values with lesser load). Finally, when the attacked nodes are back, the network changes again to a centralized topology.

In Fig. 4.(b) we see how the attack affects to the search times. As expected, those times increase to values close to those obtained at the beginning of the experiment.

Then, as nodes change their connections the topology is adapted again, lowering the average search time to a fair value. It can also be observed that the network has reached again a stable state, due in part to the fact that no node gets overloaded. If well connected nodes had become congested, then their messages queues would grow indefinitely, and so would the resulting average searches times. Finally, when the 10 nodes attacked are back, the search times gradually return to the values previous to the attack. In both experiments, the proportion of discarded searches is around 0.005%, and the proportion of failed searches is less than 0.04%.

We can conclude that DANTE$_2$ can be temporarily affected by well targeted attacks. Yet, even in a scenario where nodes that have become central are all successfully attacked at the same time, and no other nodes of the same capacity remain in the system, the network adapts again to reach another efficient state. The system is never fully shut down, because it is not dependant on any particular subset of nodes.

These experiments also show how the reconnection mechanism implemented by DANTE$_2$ avoids overloading peers, preventing the network from reaching an unstable state. If there are not enough high capacity nodes that allow to form a centralized configuration, the resulting topology becomes more 'randomized'. In conclusion, DANTE$_2$ maintains at all times the topology as clustered as possible, but at the same time prevents nodes from becoming overloaded.

DANTE$_2$ vs. GIA. As explained in Sect. 2.2, Gia is another proposal of a P2P system that uses an adaptation mechanism to improve the efficiency of searches. In [1] Gia authors carried on some simulations that show how self-adapting networks can offer a better performance than other solutions (like flooding) in a variety of scenarios. Thus, instead of repeating those same simulations with DANTE$_2$, we have deemed more interesting to compare Gia and DANTE$_2$.

We have developed a Gia simulator that implements the mechanisms described in [1]: a *flow control* system to avoid overloading nodes, a *biased random walk* search mechanism, and a *topology adaptation* protocol. An important parameter of Gia is the *maximum number of neighbors* (*max_neigh*). Nodes in Gia try to connect to as many nodes as possible, and to those with the highest capacity. We have set that limit to 20 (twice the number of *native* connections in DANTE$_2$), so that the average degree is the same as the one obtained in the DANTE$_2$ simulations. Gia advocates could reason that setting a higher maximum bound would improve performance, as searches would need less hops to locate resources. But then, it would be enough in DANTE$_2$ to increase the number of nodes $native$ connections to $max\_neigh/2$ again.

Simulations were run with 1000 nodes, with replication $r = 0.1$. As usual, nodes capacities and bandwidths are set following the distribution of Table 1. Only searches started between minutes 31 and 60 are taken into account.

In Fig. 5.(b) we plot the average search times for different loads on both systems. DANTE$_2$ seems to perform better than Gia for all loads. Additionally, beyond a certain point, Gia search times start to grow quickly with the system load, while DANTE$_2$ is able to keep search times low for the same loads. Figure 5.(a) helps us to understand the reason of DANTE$_2$'s better behavior: searches in Gia need many more hops to find a certain resource (about 160) than in DANTE$_2$ (about 7). The reason is that, although the topology in DANTE$_2$ is not totally centralized (as there are not enough high capacity

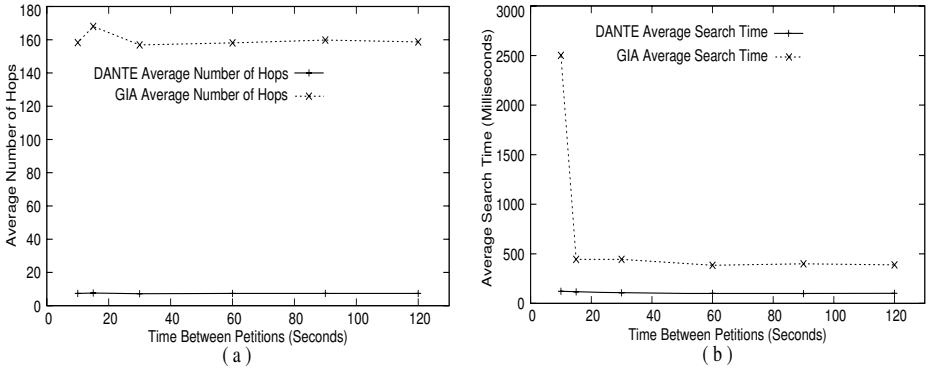


Fig. 5. DANTE and GIA searches number of hops and duration

nodes), it still keeps a clustered form where a few nodes are well connected and hence allows queries to be completed in a few hops.

All searches were successful in DANTE$_2$. Gia, on the other hand, presented a certain proportion of failed searches (between 1.5% and 2%) in all experiments.

6 Conclusions and Future Work

P2P systems are a promising new paradigm, yet they demand innovative solutions to new problems like resource location. DANTE$_2$ proposes a self-adapting mechanism that makes the network change its topology aiming always to an efficient configuration that depends on the system load and the peers capacities. The results obtained with DANTE$_2$ seem promising. However, much work remains to be done in order to improve the performance of these techniques. For example, new reconnection heuristics could be studied and developed.

References

1. Chawathe, Y., Ratnasamy, S., Lanham, N., Shenker, S.: Making Gnutella-like P2P systems scalable. In: Proceedings of the 2003 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2003), Karlsruhe, Germany (2003) 407–418
2. Lv, Q., Ratnasamy, S., Shenker, S.: Can heterogeneity make Gnutella scalable? In: Revised Papers from the First International Workshop on Peer-to-Peer Systems, Cambridge, United States (2002) 94–103
3. The napster website http://www.napster.com.
4. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2001), San Diego, CA, United States (2001) 149–160
5. Androutsellis-Theotokis, S., Spinellis, D.: A survey of peer-to-peer content distribution technologies. ACM Computing Surveys **36** (2004) 335–371

6. Ritter, J.: Why gnutella can't scale. no, really. (Technical report) Electronic format in http://www.darkridge.com/ jpr5/doc/gnutella.html.
7. The edonkey and overnet website http://www.edonkey2000.com.
8. Fletcher, G.H.L., Sheth, H.A., Borner, K.: Unstructured peer-to-peer networks: Topological properties and search performance. In: Proceedings of the Third International Workshop on Agents and Peer-to-Peer Computing, New York, New York, United States (2004) To be published by Springer.
9. Lv, Q., Cao, P., Cohen, E., Li, K., Shenker, S.: Search and replication in unstructured peer-to-peer networks. In: Proceedings of the 16th international conference on Supercomputing, New York, New York, United States (2005) 84–95
10. Cholvi, V., Laderas, V., López, L., Fernández, A.: Self-adapting network topologies in congested scenarios. Physical Review E **71** (2005) 035103
11. Adamic, L.A., Huberman, B.A., Lukose, R.M., Puniyani, A.R.: Search in power law networks. Physical Review E **64** (2001) 46135–46143
12. Gkantsidis, C., Mihail, M., Saberi, A.: Random walks in peer-to-peer networks. In: Proceedings of the Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, INFOCOM 2004. Volume 1., Hong Kong (2004) 120–130
13. Guimerà, R., Díaz-Guilera, A., Vega-Redondo, F., Cabrales, A., Arenas, A.: Optimal network topologies for local search with congestion. Physical Review Letters **89** (2002)
14. Saroiu, S., Gummadi, P.K., Gribble, S.D.: A measurement study of peer-to-peer file sharing systems. In: Proceedings of SPIE (Proceedings of Multimedia Computing and Networking 2002, MMCN'02). Volume 4673. (2002) 156–170
15. Rodero-Merino, L., López, L., Fernández, A., Cholvi, V.: Dante: A self-adapting peer-to-peer system. In: Lecture Notes in Computer Science (Proceedings of AP2PC 2006, to appear), Springer-Verlag (2006)
16. Jelasity, M., Guerraoui, R., Kermarrec, A.M., van Steen, M.: The peer sampling service: Experimental evaluation of unstructured gossip-based implementations. In: Lecture Notes in Computer Science (Proceedings of Middleware 2004). Volume 3231., Springer-Verlag (2004) 79–98
17. Newman, M.E.J.: A measure of betweenness centrality based on random walks. Social Networks **27** (2005) 39–54

A Replication-Based Fault Tolerance Protocol Using Group Communication for the Grid

Kayhan Erciyes

Izmir Institute of Technology
Computer Eng. Dept., Urla, Izmir 35430, Turkey
kayhanerciyes@iyte.edu.tr

Abstract. We describe a replication-based protocol that uses group communication for fault tolerance in the Computational Grid. The Grid is partitioned into a number of clusters and each cluster has a designated coordinator that manages the states of the replicas within its cluster. The coordinators belong to a process group and the proposed protocol ensures the correct sequence of message deliveries to the replicas by the coordinators. Any failing node of the Grid is replaced by an active replica to provide correct continuation of the operation of the application. We show the theoretical framework along with illustrations of the replication protocol and its implementation results and analyze its performance and scalability.

1 Introduction

Computational Grids consist of heterogenous computational resources, possibly with different users, and provide them with remote access to these resources [1], [2]. The Grid has attracted researchers as an alternative to supercomputers for high performance computing. One important advantage of Grid computing is the provision of resources to the users that are locally unavailable. Since there are multitude of resources in a Grid environment, convenient utilization of resources in a Grid provides improved overall system performance and decreased turnaround times for user jobs. Users of the Grid submit jobs at random times with significant turnaround times and failure of a node of the Grid would halt the execution of the application necessiating the need for fault tolerance in the Grid. Furthermore, the difficulty and the cost of recovering from faults in the Grid environment may be higher than the normal applications. Fault tolerance schemes in the Grid environment can be classified as the application specific fault tolerance based on middleware fault detection; task and data replication at middleware and transport levels; WAN, MAN and LAN resilience schemes at Internet/Network Level [3].

In this study, we propose a model and a protocol to perform task replication at middleware level for fault tolerance in the Grid using the process groups. A process group is a logical name for a set of computing elements whose membership may change with time. Replication using process groups for fault tolerance has attracted many researchers for many years [8][9][10][11]. There are several

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 672–681, 2006.
© Springer-Verlag Berlin Heidelberg 2006

systems which provide fault tolerant group communication such as Transis [7], Horus [16] and Totem [6]. Moshe [14] extends these services to a WAN. The common goal of these projects is to provide a reliable multicast communication for process groups. Total Order Multicast is the basic paradigm to provide message ordering in fault tolerant systems that use active replication [15]. It has been studied extensively and many protocols have been proposed. A detailed survey is given in [12].

An important component of the replication mechanism is the *Total Order Multicast (TOM)* protocol which ensures that all of the replicas receive the multicast messages destined to the replica group in the same order so that all of the replica finite state machines are in identical states. To achieve TOM in the Grid, we assume that the Grid is partitioned into clusters by a suitable algorithm or manually. Each cluster is controlled by a cluster head called the *coordinator*. These coordinators are the cluster heads and interface points for the ordinary nodes to the network. They perform TOM on behalf of the ordinary nodes they represent. The rest of the paper is organized as follows. Section 2 provides the background on group communication and TOM. In Section 3, the proposed protocol including the coordinator and the node algorithms is described. In Section 4, the operation of the protocol is illustrated and Section 5 provides the analysis of the algorithms. The implementation results obtained from the tests are given in Section 6 and Section 7 contains the concluding remarks along with discussions.

2 Background

2.1 Group Membership

Replication is a common approach to achieve fault tolerance in a distributed system such that replicas provide redundancy in case of a failure of a server. Two main classes of replication are the *active* and *passive* replications. In passive replication, client deals only with one replica and the primary sends messages to the secondaries to update their views. A client sends a message to all of the replicas in active replication and the states of the replicas are maintained as identical, in general, using finite state machines. To ensure consistency of the replicas, a group communication primitive called the Total Order Multicast may be used which guarantees that the requests by the clients are received by all replicas in the same order.

A group membership service manages a group of processes and is based on the *view* which is the list of processes belonging to a group. *View change* should be notified to all members. There are three basic operations needed to manage group membership effectively; *join, leave and exclude*. *Join* is executed by a process p and upon acceptance of it, all of the processes update their view. More importantly, the state of the group needs to be transferred to the new member p. A process will be removed from a group by *exclusion* if its crash is detected by a member of a group and *exit* is a voluntarily release of a process from a group by itself. The group management module should also provide the two primitives;

send_multicast to send a message to all members and *receive_multicast* to receive a message sent by a member of the group. These two primitives can be realised using various approaches such as *reliable broadcast, reliable FIFO broadcast and total order multicast. Reliable Broadcast* of a message in a group ensures that messages are delivered by all processes or none.

2.2 Total Order Multicast

Total Order Multicast (TOM) ensures that no pair of messages are delivered to the members of a group in a different order. TOM can be specified in terms of the following properties :

- *Validity* : If a correct process broadcasts a message m, then some correct process in its group will eventually deliver it.
- *Uniform Agreement* : If a process delivers a message m, then all correct processes in its group will eventually deliver it.
- *Uniform Integrity* : Every correct process in the sender's group delivers m at most once and only if m was previously broadcast.
- *Uniform Total Order* : If two correct processes deliver two messages m_1 and m_2, they do it in the same order.

Atomic broadcast is a special case of total order multicast where a TOM message is delivered to all of the group members or none. In other words, Atomic Broadcast obeys TOM and Reliable Broadcast. Atomic Broadcast or Reliable TOM protocols can be symmetric or asymmetric depending on whether some nodes are privileged in the system exist or not. Most of the symmetric protocols such as Isis [8] impose total order from the casual order relation between the messages. The static sequencer protocols such as in Amoeba [13] assume a sequencer where messages are first transmitted to this sequencer which multicasts them in order. One disadvantage of the central sequencer type of asymmetric TOM protocols is the message bottleneck around this component and having a single point of failure in the system.

3 Replication Protocol

3.1 The Model

We assume that the clusters of the Grid are already formed. For TOM in the Grid, we propose the architecture shown in Fig. 1 where replicas form clusters and each cluster is represented by a coordinator. Each replica cluster is a process group called the *replica group* and furthermore, coordinators of the clusters form a single outer group called the *coordinator group*. Election of a new coordinator is provided as in [5] if it crashes. Coordinators perform multicast communication in both groups they belong but their main function is to represent their cluster replicas in the outer coordinator group.

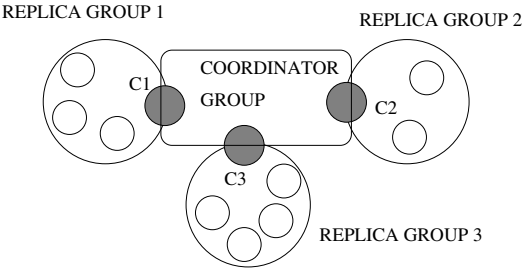


Fig. 1. The Replication Model for the Grid

3.2 The TOM Protocol

The TOM protocol proposed for the hierarhical groups use the *Static Token Algorithm (STA)* which employs similar data structures for the Token as in Suzuki-Kasami [17] algorithm for distributed mutual exclusion. A process that acquires the system wide unique token has the right to send a TOM message. The token data structure is as follows :

- Token_Sequence_Number (TSN) : integer;
- Token_Request_Queue (TRQ) : Queue of nodes;

Also every node has a *Local Sequence Number* (LSN) which shows the last sequence number of any TOM_Msg that node has received. Any node that requires to send a TOM_Msg, sends a request (Node_Req) to its coordinator as shown in the state machine diagram of Fig. 2. The request by the node is converted to a Tok_Req by the coordinator which is circulated in the ring. The coordinator sets its state to *Wait Token* (WTTK) and changes to *Hold Token* (HLTK) when it receives the token and then forwards it to the requesting node. Once a node receives the token from the coordinator, it incerements TSN and stamps the TOM message with this TSN and sends it to the coordinator. Since there is a unique Token with a unique TSN, any TOM message from any node will have a unique sequence number which provides the total ordering of the messages. The FSM of the coordinator also depicts the sequence for atomicity. When the coordinator receives the TOM_Msg from the node, it broadcasts this to the ring and upon reply, it sends TOM_Chk message to check the acknowledgements. If all nodes in the group have received the TOM_msg, the operation is succesful and a final TOM_Set message is sent to all coordinators to allow the final delivery to the nodes. Any node receiving the TOM_Set message checks whether LSN+1=TSN, that is, whether this message has arrived in sequence. If so, TOM_Msg is delivered to the application, otherwise it is delayed until prior messages arrive. At any state, a coordinator may receive a remote TOM_Msg which is not shown in Fig.2 for simplicity. In this case, the coordinator braodcasts the TOM_msg in its cluster and also receives replies from each node.

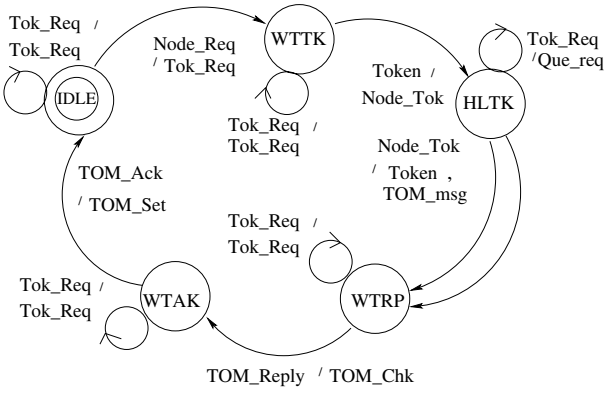


Fig. 2. The Coordinator for Static Token Algorithm

4 Illustration of STA

Fig. 3 shows an example scenario for STA. Initially, Token is held in the replica cluster 2 by coordinator C_2. For simplicity, it is assumed that coordinators are directly connected to the nodes and to each other. The following is the sequence of events :

1. Node n_{13} in cluster 1 requests Token from its coordinator C_1 by $Node\_Req$.
2. C_1, does not have the Token, hence broadcasts this request by R_{13} in the coordinator group.
3. C_2 has the token which is not being used and has TSN as 0 and its queue is empty. C_2 sets the destination of the token as n_{13} and sends this token to C_1.
4. C_1 receives the Token and sends it to n_{13} which is received by it. These first four steps are depicted in Fig. 3.(a).
5. While n_{13} is holding the Token, nodes n_{21} and n_{32} in clusters 2 and 3 make requests consecutively to their coordinators for the Token which in turn send requests R_{21} and R_{32} tothe coordinator group.
6. R_{21} and then R_{32} reach C_1 consecutively. C_1 queues these requests.
7. When n_{13} receives the token, it increments TSN of the Token to 1 and sends TOM message with this sequence number to C_1 along with the Token. Steps 5-6-7 are depicted in Fig. 3.(b).
8. C_1 receives the Token and the TOM message. It broadcasts TOM on the coordinator group. C_1 also checks its local Coordinator Token Request Queue (CRQ). It appends the nodes in its local queue to the Token Queue, removes the first node from the TRQ (n_{31}) and sends the token to C_2.
9. C_3 and C_2, pass an acknowledgement message (TOM_Ack) to the source. Steps 8 and 9 are depicted in Fig. 3.(c).
10. If C_1 receives (TOM_Ack) acknowledgements from every node in the group, the TOM is succesful. In this case, C_1 issues a TOM_Set message to finally

initate the actual delivery of the TOM message to the application. The replica node however, checks its LSN with TSN to conclude TOM delivery as described above.

11. When C_2 receives the Token from C_1, it proceeds similar to 7-8-9-10 above and when n_{21} finishes with the Token, C_2 sends it to C_3. Steps 10 and 11 are depicted in Fig. 3.(d). Note that this is performed in parallel with the TOM_Msg delivery of n_{13}.

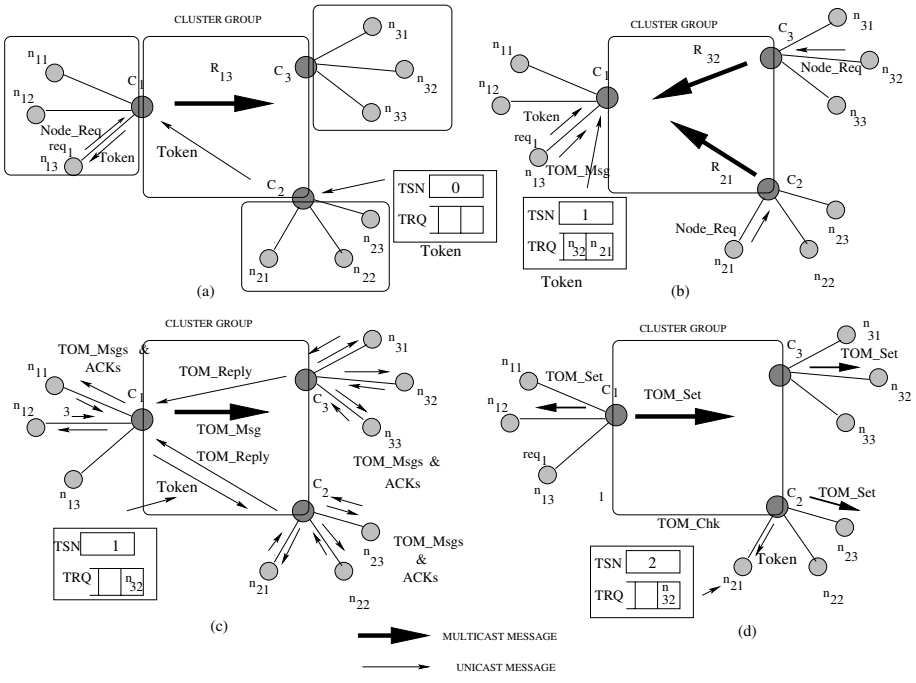


Fig. 3. Operation of the STA Algorithm

5 Analysis of STA

Assuming k, m, n and d are upperbounds on the number of clusters, nodes in a cluster in the network, nodes in the ring of coordinators and the diameter of a cluster respectively, the sending and atomically reception of the TOM message by all nodes in the group requires the following steps :

1. Request by the node : $O(d)$
2. Circulation of Tok_Req and reception of token : $O(k)$ as this is *All to All Communication* in a unidirectional ring or again $O(k)$ this is k unicast messages to implement a multicast message in case of a different architecture.
3. Sending of Token to the Node : $O(d)$

4. Sending of TOM message by the node to the coordinator : $O(d)$
5. Circulation of TOM message by the coordinator : $O(k)$
6. Local broadcast of TOM by each coordinator in its cluster $O(m)$
7. Collection of the acknowledgements from the nodes by each coordinator : $O(m)$
8. Circulation of acknowledgement message (TOM_Ack) by the coordinator : $O(k)$
9. Set operation by the source cordinator for atomicity : $O(k)$
10. Set operation by the coordinators in local clusters for atomicity : $O(m)$

Theorem 1. *The total time per TOM using STA Algorithm is* $O_{TOMT}(m)$

Proof. The total time required for TOM delivery is the sum of all of the 9 steps above which can be evaluated as follows

$$O_{TOMT} = 4k + 3m + 3d = O_{TOMT}(m) \tag{1}$$

assuming k=m and d is negligible.

Theorem 2. *The total number of messages per TOM using STA Algorithm is* $O_{TOMM}(m^2)$

Proof. The total number of messages required for TOM delivery can be found similarly by calculating the sum of messages in transit at each step of operation above except for local broadcast operations (steps 6,7 and 10) where the number of messages sent are k*m.

$$O_{TOMM} = 4k + 3km + 3d = O_{TOMM}(m^2) \tag{2}$$

assuming k=m and d is negligible.

Corollary 1. *For a network of* N *nodes, the total time per TOM using STA Algorithm is* $O_{TOMT}(\sqrt{N})$ *and the total number of messages required per TOM using STA is* $O_{TOMT}(N)$.

Proof. It was shown by theorems 2 and 3 that $O_{TOMT}(m)$ and $O_{TOMM}(m^2)$. Since total number of nodes in a network N is equal to the total number of nodes in the model network which is km=m^2 assuming k=m, $O_{TOMT}(\sqrt{N})$ and $O_{TOMT}(N)$.

An algorithm like Suzuki-Kasami [17] would require 0 or N messages ($N-1$ for requests and 1 for Token). By Corollary 1, we can conclude that STA provides and order of magnitude decrease in TOM execution time including obtaining the Token with respect to an algorithm like Suzuki-Kasami.

5.1 Verification of Total Order Multicast Properties by STA

The TOM properties can be verified for STA operation follows :

- *Validity* : When a correct process broadcasts a message *msg*, then all of the correct *TO_Node* processes will deliver this message as directed by their cluster coordinators in TOM_Set message.
- *Uniform Agreement* : The delivery of the messages to the application by all of the *TO_Node* processes occur in the same cycle and the messages are delivered by all of the correct *TO_Node* processes in the cluster identities of which are supervised by the local coordinator.
- *Uniform Integrity* : The messages at a node are delivered at most once by the TOM_Set message issued by the coordinator.
- *Uniform Total Order* : The messages are delivered in the same order as there is only one sequence number (TSN) kept at Token and only the holder of the Token can send a TOM message. Each node has a local sequence number (LSN) and will not deliver a message that is larger than its LSN+1 which means any out of order messages wil be delayed until the TOM_Msg with the correct sequence arrives.

6 Implementation Results Using MPI

We implemented the architecture shown in Fig. 1 using the MPI (Message Passing Interface) [4] over a cluster of 20 processors. The end-to-end run times of the STA Algorithm for a single request, from the time of the request until the actual delivery of the TOM message to the application are measured against varying number of clusters and the size of clusters as shown in Fig. 4.

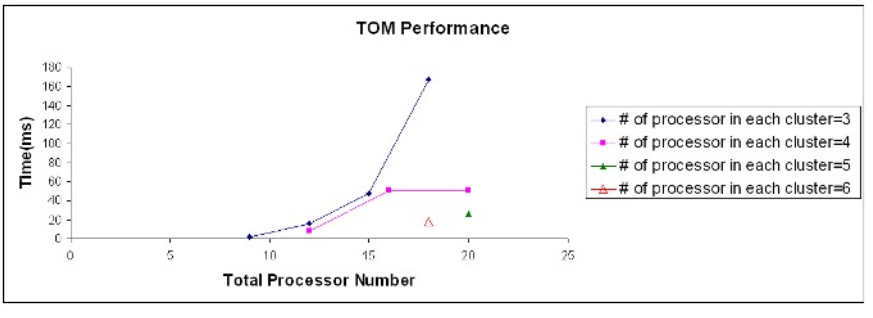


Fig. 4. TOM Run Time Results against Cluster Size and Numbers

MPI multicast message facility was used for group communication within a cluster. As shown in the figure, the run time of STA increases linearly with the number of clusters and also the count of processors in each cluster. The performance of the proposed architecture in terms of the size of the single message delivered to the application is depicted in Fig. 5. It can be observed that the delivery times are almost stable with respect to cluster numbers and sizes.

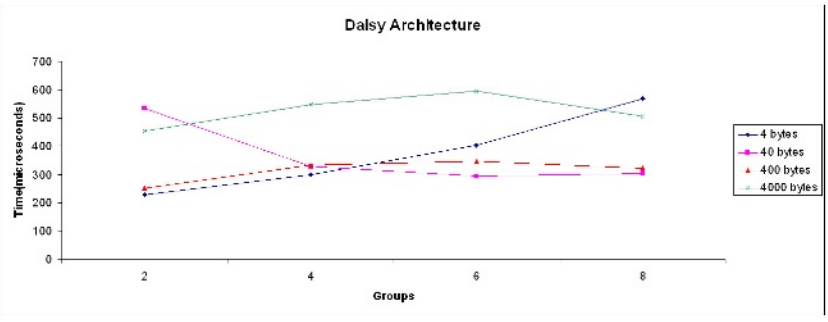


Fig. 5. Daisy Architecture Run Time Results against Message Size

7 Discussions and Conclusions

We proposed a framework and a protocol with two algorithms to implement TOM in the Grid to provide fault tolerance by replication to the application. We showed that the proposed algorithms provide significant gains in the number of messages and the time to deliver TOM messages theoretically with respect to a flat architecture without any hierarchies. The preliminary results indicated that the protocol is scalable in terms of run times and the sizes of the messages delivered. The coordinators have an important role and they may fail. New coordinators may be elected and any failed node member can be excluded from the cluster which is an improvement over classical algorithms as they do not provide recovery for a crashed node in general. The recovery procedures can be implemented using algorithms as in [5] which is not discussed here. One other advantage of the proposed model is the pre-processing of the requests of the nodes by the coordinators are performed independently resulting in improved performance. We are looking into implementing this protocol in a Grid environment with larger cluster sizes and counts and measure the performance of the whole protocol including the clustering algorithm.

Acknowledgements

I would like to thank Orhan Dagdeviren for his help in the experimental setup of the tests in Section 6.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal of High Performance Computing Applications, 15(3), (2001), 200-222.
2. Foster, I.: What is the Grid ? A Three Point Checklist, Grid Today, 1(6), (2002).

3. Valcarenghi, L et al. : QoS Aware Fault Tolerance in Grid Computing, Workshop on Reliability and Robustness in Grid Computing Systems, GGF16, Feb. 13-16, 2006, Athens, Greece.
4. MPICH-G2: A Grid-enabled Implementation of the Message Passing Interface, Journal of Parallel and Distributed Computing, 63(5), (2003), 551 - 563.
5. Tunali, T, Erciyes,K., Soysert, Z.: A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System, Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems, 2(1), (2000), 33-44.
6. Y.Amir et all, The TOTEM Single Ring Ordering and membership Protocol, ACM Trans. Comp. Systems., 13(4),1995.
7. Y.Amir et all, Transis: A communication subsystem for high availability. Proc. of 22nd IEEE Int'l Symp. on Fault-Tolerant Computing, IEEE Press, NJ, pp. 76-84.
8. Birman K. P., van Renesse, R., Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, Los Alamitos, Ca., 1994.
9. Birman K. P., The Process Group Approach to Reliable Distributed Computing, Communications of the ACM, 36(12), December 1993.
10. Chockler, G, Keidar, I., Vitenberg, R., Group communication specifications: a comprehensive study, ACM Computing Surveys, 33 (4), December 2001, pp. 427 - 469.
11. Cristian F., Synchronous and Asynchronous Communication, Communications of the ACM. Special Section on Group Communication, 39(4), April 1996.
12. Defago, X., Agreement Related Problem : From semi-passive replication to Totally Ordered Broadcast. Ph.D. thesis, Ecole Polytechnique Lausanne, Switzerland, Aug. 2000.
13. Kaashoek, M. F., Tanenbaum, A. S., Group Communication in the Amoeba distributed operating system, Proc. of the 11th IEEE International Conf. on Distributed Computing Systems, IEEE Computer Society press, pages 436-447
14. Keidar, I. et al, Moshe: A group membership service for WANs, ACM Transactions on Computer Systems (TOCS) 20 (3) , August 2002, 191-238.
15. Schenider, F., Replication management using the state-machine approach, *Duistributed Systems*, ACM Press, 169-198.
16. Van Renesse R., Birman K. P.,Maffeis S., Horus : A Flexible Group communication System, Communications of the ACM, Special section on Group Communication, 39(4), April 1996
17. Susuki, I., Kasami, T. : A Distributed Mutual Exclusion Algorithm, ACM Trans. Computer Systems, Vol. 3(4), (1985), 344-349

Increasing Availability in a Replicated Partitionable Distributed Object System*

Stefan Beyer[1], Mari-Carmen Bañuls[2], Pablo Galdámez[1], Johannes Osrael[3], and Francesc D. Muñoz-Escoí[1]

[1] Instituto Tecnológico de Informática, Universidad Politécnica de Valencia, Camino de Vera, s/n, 46022 Valencia, Spain
{stefan, pgaldamez, fmunyoz}@iti.upv.es
[2] Max-Planck-Institut fuer Quantenoptik, Hans-Kopfermann-Str. 1, D-85748 Garching, Germany
banulsm@rzg.mpg.de
[3] Vienna University of Technology, Institute of Information Systems, Distributed Systems Group, Argentinierstrasse 8/184-1, 1040 Vienna, Austria
j.osrael@infosys.tuwien.ac.at

Abstract. Replicating objects in distributed object systems provides fault-tolerance and increases availability. We have designed a replication protocol for distributed object systems that provides increased availability by relaxing consistency temporarily. The protocol allows all partitions in a partitioned system to continue operating. The states of certain replicas are allowed to diverge. The application programmer can specify the required consistency using integrity constraints.

We present an analytical model of the new protocol and evaluate it against the primary partition model, where only a majority partition is allowed to continue. Furthermore, we identify the type of application for which our protocol provides increased availability.

1 Introduction

Replication is a common means of providing fault-tolerance and improving availability in distributed systems. We propose a new replication protocol for distributed object systems, which allows availability to be increased by temporarily relaxing consistency. The protocol enables nodes in a partitioned system to continue operating, in contrast to the primary partition model [1], where only one partition can continue. An analytical model of the protocol is presented in this paper and compared with the primary partition model.

The new replication protocol forms part of the DeDiSys middleware architecture [2]. The aim of the DeDiSys research project is to investigate the trade-off between consistency and availability. Furthermore, the objective is to allow this trade-off to be configured. The main idea is that by reducing consistency temporarily, availability can be improved.

* This work has been funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract number 004152).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 682–695, 2006.
© Springer-Verlag Berlin Heidelberg 2006

A partitionable system is a system in which groups of nodes or individual nodes might become separated from the rest of the system, because of a network failure. Replicating objects in such a system allows operations on objects to continue, even if a node that hosts an object becomes unreachable, as long as a replica remains accessible. The degree to which a system can continue operating on a replica depends on the policy used for keeping replicas up-to-date and on the consistency required by the application. Object versions in various partitions might diverge, since conflicting accesses to replicas of the same object in different partitions can not be detected until the network failure is repaired and the system has recovered. If the required consistency is strong, operations must be severely limited during partitioning. These limitations reduce the availability of the system.

We base consistency on integrity constraints. Each constraint is associated with an operation and defined over parts of the state of one or more objects. An example of such a constraint would be a rule that states no single person can reserve more than ten tickets in a ticket booking system. Such a constraint can be coded easily in a constraint object in our system. Constraints are described in more detail in [2]. The system is said to be fully **constraint consistent**, if only changes to objects caused by operations whose constraints have been met are accepted.

During partitioning we allow consistency to be relaxed. However, once the system has recovered, it has to be in a fully constraint-consistent state. To this end, we introduce a classification of integrity constraints, reflecting different degrees of consistency.

We have obtained initial performance results for the new protocol, by evaluating it in the DeDiSys Lite prototype environment, which serves as an environment to simulate replication protocols. These results are presented in [3].

2 Related Work

Mobile databases provide a variety of solutions to the partitioning problem. In these systems reconciliation between replicas whose states have diverged is a common occurrence.

Bayou [4] allows temporarily disconnected nodes to synchronise efficiently using an anti-entropy protocol. Write-write conflicts are dealt with in an application-specific manner. Although this approach is very flexible, it imposes the need to write complex conflict resolution procedures on the application programmer.

Tentative transactions [5] are used by many systems to allow transactions to commit locally. At reconnection time it is attempted to commit the previously locally committed transactions at a server. The locally committed changes have to be undone, if the server commit fails.

The mobile transactions in [6] extend the concept of tentative transactions to allow the application to specify pre-conditions and post-conditions. The authors of [7] apply this principle to a distributed object system.

Phatak and Badrinath [8] keep track of different versions of data-items, in order to resolve conflicts by "agreeing" on a non-conflicting past version of replicas.

Finally, a technique used to reduce conflicts is to restrict operations to commutative operations as much as possible [9] [5].

The replication framework presented in this paper uses some of the techniques used in mobile database systems and applies them to a distributed object model. In particular, a form of tentative transactions with multi-versioning is used.

The trade-off between consistency and availability has been investigated in the context of other distributed systems. These systems generally require the application programmer to specify the required consistency or the required availability.

The authors of [10] use consistency units (conits) to specify the bounds on allowed inconsistency. A conit is a set of three values representing "numerical error", "order error" and "staleness". The system does not support partitioning.

In CoRe [11] the principle of specifying consistency is extended to allow the programmer to define consistency using a larger set of parameters. The system only focuses on data objects; that is, objects that do not cause invocations to other objects.

AQua [12] approaches the trade-off from the other direction by allowing availability requirements to be specified. In AQua "quality objects" are used to specify quality of service requirements. AQua considers crash failures, value faults and time faults, but does not consider partitioning.

None of the distributed systems described above considers constraint consistency.

3 System Model

We assume a partially synchronous system. In such a system clocks are not synchronised, but message time can be bound. We also assume the presence of a group membership service [13] which provides all the server nodes with a single view of which nodes are part of the system or the current partition. Furthermore, a group communication service provides the server nodes with reliable FIFO broadcast according to the definition in [14].

The "pause-crash model" [15] is assumed for node failures, and the "link failure model" [16] for communication services. As we cannot distinguish between a failed node and an isolated node until recovery time, we treat every failure as partitioning. Partitions can occur in any number and order. Recovery of partitioning can be in a different order in which the partitioning originally occurred.

We employ a relaxed *passive replication* model. In passive replication [17] [18] requests are only processed by one *primary copy*. Updates are then propagated to the *secondary copies*. However, we do relax this for read-only operations. Read-only operations can be served by any secondary copy.

If a primary copy of an object is not reachable, a secondary copy is promoted to a temporary primary copy. We therefore, have a "primary per partition model".

4 The Primary-Per-Partition Protocol

We have developed a replication protocol that allows all partitions to continue, when groups of nodes are separated due to a network failure. The Primary-per-Partition Protocol (P4) relaxes consistency during partitioning, but full consistency is restored at reconciliation time. If the primary copy of the object is in a different partition, a secondary copy is promoted to a temporary primary. In order to increase system availability, we allow write operations on temporary primaries in certain conditions.

Consistency is based on integrity constraints. Constraints are associated with object methods. They restrict the state the objects have to be in for an operation to be executed (*pre-condition*). Alternatively, they restrict the state in which the objects have to be in after an operation is executed (*post-condition*). In order to evaluate a constraint reliably, an up-to-date version of all objects that participate in the constraint is needed. During partitioning however, secondary copies of an object might be stale, if the primary copy resides in a different partition. During reconciliation constraints might be violated retrospectively, when missed updates are propagated. Therefore, some operations that where performed during partitioning might have to be undone to restore consistency. This behaviour might be acceptable for the majority of the operations, but there are some operations that should never occur, if they might have to be undone later on.

We therefore introduce the notion of **critical constraints**. A constraint labelled critical is a constraint that needs up-to-date versions of all of the participating objects. Such a constraint cannot be evaluated if a participating object is stale. Furthermore, the protocol has to ensure that critical constraints are never violated "in retrospect" during conflict resolution. In contrast **regular constraints** can be evaluated on stale objects. A post-condition expressed as a regular constraint has to be re-evaluated, once all the objects are up-to-date. However, a pre-condition is not re-evaluated at reconciliation time.

Read operations cannot introduce data inconsistencies. The following description therefore focuses on write operations.

Normal Mode

1. All object write invocations have to be directed to the primary replica.
3. All the pre-condition constraints, associated with the operation are evaluated. If a constraint is not met, the operation is aborted.
4. The operation is invoked. Nested invocations might be started.
5. Once the primary replica has updated its local state, all the post-condition constraints, associated with the operation are evaluated. If a constraint is not met, the operation is aborted.
6. Once these checks have been successfully completed, all primary replicas updated in the operation propagate the new object states to the backup replicas.
8. Once this update transfer has terminated, the operation result is returned to the client.

Degraded Mode
A write operation in degraded mode is similar to that in normal mode with the following additions:

1. If the primary copy of an object being written to is not found, a secondary copy is chosen in some pre-determined way, for example based on the replica identifier. The chosen secondary replica is promoted to a "temporary primary". This is not done, if the operation has a critical constraint as a pre- or post-condition.
2. Objects that are changed are marked as "revocable", if any of the post-condition constraints associated to the operation that has been executed has been evaluated on possibly stale objects.
3. Critical constraints are not evaluated, if a participating object might be stale. If this were the case, the operation is aborted.
4. Regular post-condition constraints with possibly stale objects are marked for re-evaluation at reconciliation time.
5. Operations with critical constraints that include a revocable object are not permitted, so that critical constraints cannot be violated retrospectively, when a revocable object is rolled back.

Reconciliation
When two or more partitions re-join, reconciliation is started. During this process no write operations are processed. Reconciliation is done in three phases:

Phase 1: Restoring replica consistency. When partitions are being joined, replica consistency is restored. If two primary copies of the same object have been modified in different partitions a write-write conflict has occurred. To solve this conflict the application is asked to resolve the conflict. To this end a handler routine which has been previously registered by the application is called. Conflict resolution strategies the application may employ range from choosing one of the conflicting primary copies to installing a completely new version.
Phase 2: Restoring constraint consistency. All constraints that are marked for re-evaluation and for which the original primary copy of all participating objects is now available are now re-evaluated. If a constraint is violated, the application is again asked to resolve the conflict. To this end another handler routine is called. The application handler can restore consistency by setting one of the objects marked as revocable to a state that meets the constraint. All other post-condition constraints of operations that have been executed during partitioning and in which the revocable object participates have to be re-evaluated. This re-evaluation has to be performed to avoid constraints being violated retrospectively.
Phase 3: Updating secondary copies. Finally, all changes to primary copies which have occurred in phase 1 or phase 2 have to be applied to the secondary copies of the modified objects.

4.1 Automatic Reconciliation

The P4 protocol can also be employed without application interaction; that is, through using automatic reconciliation, instead of application handler routines. However this involves storing a large amount of extra data about object changes during degraded mode. For each object, a list with previous versions has to be kept. Furthermore, for each of the object versions a list of the nodes present in the current partition during the time of the write access needs to be kept. Finally, for each regular post-condition constraint, a reference to the last known version to meet the constraint of each updated participating object needs to be saved.

If several partitions try to re-join and write-write conflicts between two primary copies occur, the protocol can choose one of the primaries according through some pre-defined precedence order or a more complex algorithm.

Constraint violations that are detected at reconciliation time could also be dealt with automatically. If on re-evaluation a regular constraint is not met, one associated object marked as revocable is chosen to revert to its previous version. Among all the revocable objects, one of them is chosen following an increasing order of object identifiers. This object is reverted to previous versions repeatedly, until a version is found that either leads to the regular consistency constraint being met or has the same version number as the last known version that satisfied the constraint. If the later case occurs, without the constraint being met, another revocable object must go through the process of reverting to previous versions. Once the constraint is met, all other regular post-condition consistency constraints associated with the objects that have reverted to a previous version have to be re-evaluated.

5 Availability Model

5.1 Overview

An analytical availability model has been developed for the new protocol described in section 4. A model for the primary partition model [1] has also been developed, in order to compare the availability of the two approaches. The primary partition model has been traditionally used when strong consistency is a requirement. The model only allows a majority partition to proceed in the case of partitioning. Such a majority partition has to contain more than half of the nodes in the system. The primary partition model has the advantage that secondary copies in the proceeding partition are never stale, as no changes can occur in other partitions. Hence, the availability in one partition can theoretically be higher than in our model, but is zero in all other partitions. The P4 protocol in contrast suffers from possibly stale replicas, but allows every partition to proceed. Intuitively, the P4 appears to provide better availability in applications with more regular than critical constraints.

5.2 Parameters

The independent parameters that determine the behaviour of the model are the following:

Related to the system:

• $N \rightarrow$ Number of nodes in the system.
• $\{P_1, P_2, \ldots P_q\} \rightarrow$ Partitioning pattern (q is the total number of partitions, and P_i the number of nodes of the i-th one). P_M is the cardinality of the majority partition (if there is no majority, $P_\mathrm{M} = 0$, otherwise $P_\mathrm{M} > \frac{N}{2}$).

Related to the data:

• $M \rightarrow$ Total number of objects.
• $r \rightarrow$ Replication degree (number of replicas per object).
• $R_\mathrm{c} \rightarrow$ Number of critical constraints.
• $o_\mathrm{c} \rightarrow$ Average number of objects involved in a critical constraint.
• $R_\mathrm{nc} \rightarrow$ Number of regular constraints.
• $o_\mathrm{nc} \rightarrow$ Average number of objects involved in a regular constraint.

As an alternative to R_c and R_nc we may use the pair of parameters $R_T \equiv R_\mathrm{c}+R_\mathrm{nc}$ and $\rho \equiv \frac{R_\mathrm{c}}{R_\mathrm{c}+R_\mathrm{nc}}$.

Related to the application:

• $\tau_\mathrm{I} \rightarrow$ Number of invocations launched per node and time unit.
• $f \rightarrow$ Ratio of writing invocations to the total.
• $n \rightarrow$ Depth of an invocation, i.e. order of nesting.
• $o_\mathrm{I} \rightarrow$ Width of the invocation, i.e. average number of objects involved in a single level.

5.3 Definitions and Assumptions

We calculate the following quantities:

• $\mathrm{NA_R(PP)} \rightarrow$ Non–availability for reading operations in the *primary partition* (PP) model.
• $\mathrm{NA_W(PP)} \rightarrow$ Non–availability for writing operations in the *primary partition* model.
• $\mathrm{NA_R(P4)} \rightarrow$ Non–availability for reading operations in the *primary per partition* (P4) model.
• $\mathrm{NA_W(P4)} \rightarrow$ Non–availability for writing operations in the *primary per partition* model.

Each of them is defined as the ratio of the failed operations, i.e. those that cannot be completed due to non-availability of some resource, to the total number of invocations of the same type, $\tau_\mathrm{I} f N$ and $\tau_\mathrm{I}(1-f)N$ for writing and reading invocations, respectively.

The following assumptions have been identified and are used in the model:

• In the *primary partition* model (PP), no writing operation is allowed to proceed if launched in a minority partition.
• Reading operations are always possible, unless the current partition contains no replica of a required object.
• Finally, the distribution of replicas and primaries among nodes, and that of

objects into constraints are all assumed to be uniform. Moreover, the constraints are *sparse*, so that the probability of a certain object to enter more than one constraint is neglected. This can only be true if $o_{\rm c}R_{\rm c} + o_{\rm nc}R_{\rm nc} \lesssim M$.

5.4 Some Probabilities and Other Useful Quantities

Here we deduce or mention some quantities which are extensively used in the following sections:

- Probability of a given node to hold a replica of a given object $\rightarrow \frac{r}{N}$.
- Probability of a given object to be part of a critical (regular) constraint $\rightarrow \frac{o_{\rm c}R_{\rm c}}{M}$ $\left(\frac{o_{\rm nc}R_{\rm nc}}{M}\right)$. This must be small (or at least ≤ 1), according to the *sparseness* hypothesis.
- Probability of a certain node to hold the original primary of a given (fixed) object $\rightarrow \frac{1}{N}$.

Thus, the probability of having the primary within partition $P_{\rm i}$ is $\frac{P_{\rm i}}{N}$.

- Total number of objects directly accessed by an invocation $\rightarrow o_{\rm D} \equiv \sum_{k=0}^{n} o_{\rm I}^k = \frac{o_{\rm I}^{n+1} - o_{\rm I}}{o_{\rm I} - 1} + 1$.
- Total number of objects accessed by an invocation (including those that are only used for checking constraints) $\rightarrow$

$$\tilde{o}_{\rm I} \equiv o_{\rm D} + o_{\rm constraints} = o_{\rm D}\left[1 + \frac{o_{\rm c}R_{\rm c}}{M}\left(o_{\rm c} - 1\right) + \frac{o_{\rm nc}R_{\rm nc}}{M}\left(o_{\rm nc} - 1\right)\right].$$

- Probability of not having a replica of a certain object in partition $P_{\rm i} \rightarrow \left(1 - \frac{r}{N}\right)^{P_{\rm i}}$.
- Probability that an invocation involves no critical constraints $\rightarrow \left(1 - \frac{o_{\rm c}R_{\rm c}}{M}\right)^{o_{\rm D}}$.

5.5 Primary Partition Model

Writing invocations In the *primary partition* model, a writing operation fails if either it is launched in a minority partition or it starts in the majority, but fails to find an available replica of some object accessed by the invocation directly or through the evaluation of a constraint. Then we may write

$$\mathrm{NA_W(PP)} = \sum_{i, P_{\rm i} \leq N/2} \frac{P_{\rm i}}{N} + \frac{P_{\rm M}}{N}\left[1 - \left(1 - \left(1 - \frac{r}{N}\right)^{P_{\rm M}}\right)^{\tilde{o}_{\rm I}}\right]. \tag{1}$$

The subexpression between brackets represents the complementary of the probability of having at least one replica of every accessed object $\tilde{o}_{\rm I}$ within the majority partition. Such term must vanish in the case of total replication, $r = N$. In that case, we may check that the non-availability equals the proportion of nodes that are not in a majority partition,

$$\left(\mathrm{NA_W(PP)}\right)_{\rm r=N} = \frac{1}{N}\sum_{\rm minority} P_{\rm i} = 1 - \frac{P_{\rm M}}{N}. \tag{2}$$

If no majority partition exists ($P_{\rm M} = 0$), there is no availability at all in this model.

Reading invocations Since reading is always allowed on stale copies of any object, the only source of non-availability for reading invocations is the lack of replicas of the required objects in the current partition.

$$\mathrm{NA_R(PP)} = \sum_{i=1}^{q} \frac{P_\mathrm{i}}{N} \left[1 - \left(1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right)^{o_\mathrm{D}} \right], \tag{3}$$

where the sum is taken over all partitions.

Differently to the writing case, here the only replicas required are those of directly accessed objects, as no constraint needs to be checked for a reading invocation.

5.6 Primary Per Partition Model

Writing invocations. In the *primary per partition* model, there is no difference between the behaviour in minority and majority partitions. A writing operation always fails if there is at least one "accessed" object that has no replica available in the local partition, but due to the existence of critical constraints and revocable objects that might roll-back to a former state at reconciliation time, an invocation may fail even having all the replicas available. In particular, the invocation will fail if the original primary is missing for some of its critical objects. By critical object of an invocation we mean an object that is subject to a critical constraint, and which is either directly accessed by the invocation or whose critical constraint affects one of the directly involved objects. The probability of involving some critical constraint can be written

$$\left[1 - \left(1 - \frac{o_\mathrm{c} R_\mathrm{c}}{M} \right)^{o_\mathrm{D}} \right].$$

If we define $\tilde{o}_\mathrm{c}$ to be the total number of critical objects accessed by the invocation, $\tilde{o}_\mathrm{c} = o_\mathrm{D} \frac{o_\mathrm{c}^2 R_\mathrm{c}}{M}$, the probability of missing some of the original primaries in the partition P_i reads

$$\left(1 - \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c}} \right).$$

There is still one source of non-availability to be taken into account. Even if all the original primaries for critical objects are available in the current partition, if any of them has already been written by some earlier invocation since the partition occurred, the evaluation of the corresponding constraint will not be possible, and the invocation will also fail. We may define $P_\mathrm{W}^{(\mathrm{c})}(P_\mathrm{i}, t)$ to be the probability of a critical object to have been written before instant t in the partition P_i. This will depend on the frequency of invocations, τ_I, f, the other system parameters and the time since the partition occurred. In terms of this probability, we may write

$$\mathrm{NA_W(P4)} = \sum_{i=1}^{q} \frac{P_\mathrm{i}}{N} \left\{ \left[1 - \left(1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right)^{\tilde{o}_\mathrm{I}} \right] + \left[1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right]^{\tilde{o}_\mathrm{I}} \left[1 - \left(1 - \frac{o_\mathrm{c} R_\mathrm{c}}{M} \right)^{o_\mathrm{D}} \right] \left[\left(1 - \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c}} \right) + \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c}} \left(1 - \left(1 - P_\mathrm{W}^{(\mathrm{c})}(P_\mathrm{i}, t) \right)^{\tilde{o}_\mathrm{c}} \right) \right] \right\}. \quad (4)$$

Or, in a slightly more compact manner,

$$\mathrm{NA_W(P4)} = \sum_{i=1}^{q} \frac{P_\mathrm{i}}{N} \left\{ \left[1 - \left(1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right)^{\tilde{o}_\mathrm{I}} \right] + \left[1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right]^{\tilde{o}_\mathrm{I}} \left[1 - \left(1 - \frac{o_\mathrm{c} R_\mathrm{c}}{M} \right)^{o_\mathrm{D}} \right] \times \left[1 - \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c}} \left(1 - P_\mathrm{W}^{(\mathrm{c})}(P_\mathrm{i}, t) \right)^{\tilde{o}_\mathrm{c}} \right] \right\}. \quad (5)$$

We may calculate $P_\mathrm{W}^{(c)}(P_\mathrm{i}, t)$ as the number of critical objects that have been written between $t = 0$ and t divided by the number of critical objects that reside in partition P_i ($o_\mathrm{c} R_\mathrm{c} P_\mathrm{i}/N$). The increment in the number of already written critical objects in the time interval $[t, t + dt]$ is determined by the number of writing invocations that involve some critical constraint and whose primaries are available and have not yet been changed since $t = 0$. Each such successful invocation modifies $\tilde{o}_\mathrm{c}/o_\mathrm{c}$ critical objects.[1] Then the number of critical objects written from t to $t + dt$ is

$$\tau_\mathrm{I} P_\mathrm{i} f \left[1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right]^{\tilde{o}_\mathrm{I}} \left[1 - \left(1 - \frac{o_\mathrm{c} R_\mathrm{c}}{M} \right)^{o_\mathrm{D}} \right] \times \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c}} \left(1 - P_\mathrm{W}^{(c)}(P_\mathrm{i}, t) \right)^{\tilde{o}_\mathrm{c}} \frac{\tilde{o}_\mathrm{c}}{o_\mathrm{c}} dt. \quad (6)$$

This allows us to write a differential equation for $P_\mathrm{W}^{(c)}(P_\mathrm{i}, t)$

$$\frac{dP_\mathrm{W}^{(c)}(P_\mathrm{i}, t)}{dt} = B_\mathrm{I}^{(c)}(P_\mathrm{i}) \left(1 - P_\mathrm{W}^{(c)}(P_\mathrm{i}, t) \right)^{\tilde{o}_\mathrm{c}}, \quad (7)$$

where we have defined the auxiliary function

$$B_\mathrm{I}^{(c)}(P_\mathrm{i}) \equiv \tau_\mathrm{I} P_\mathrm{i} f \left[1 - \left(1 - \frac{r}{N} \right)^{P_\mathrm{i}} \right]^{\tilde{o}_\mathrm{I}} \left[1 - \left(1 - \frac{o_\mathrm{c} R_\mathrm{c}}{M} \right)^{o_\mathrm{D}} \right] \left(\frac{P_\mathrm{i}}{N} \right)^{\tilde{o}_\mathrm{c} - 1} \frac{o_\mathrm{D}}{M}.$$

[1] As each object is assumed to take part in a single constraint, the invocation may only write one object per constraint checked.

The equation above is separable and can be easily solved (integrate with the change of variable $y = 1 - P_{\mathrm{W}}^{(c)}$, $\frac{dy}{dt} = -\frac{dP_{\mathrm{W}}^{(c)}}{dt}$).

After imposing the initial condition $P_{\mathrm{W}}^{(c)}(P_{\mathrm{i}}, 0) = 0$, the expression for $P_{\mathrm{W}}^{(c)}$ reads

$$P_{\mathrm{W}}^{(c)}(P_{\mathrm{i}}, t) = 1 - \left[1 - (1 - \tilde{o}_{\mathrm{c}})\, B_{\mathrm{I}}^{(c)}(P_{\mathrm{i}})\, t\right]^{\frac{1}{1-\tilde{o}_{\mathrm{c}}}}, \tag{8}$$

valid for $\tilde{o}_{\mathrm{c}} \neq 1$. If $\tilde{o}_{\mathrm{c}} > 1$, $P_{\mathrm{W}}^{(c)}$ is a monotonically increasing function of t, well defined over $t \in [0, \infty[$, which approaches 1 as $t \to \infty$. If $\tilde{o}_{\mathrm{c}} < 1$, the solution of the equation is well defined for $t < t_{\max} \equiv \frac{1}{B_{\mathrm{W}}^{(c)}(P_{\mathrm{i}})(1-\tilde{o}_{\mathrm{c}})}$, whereas for $t > t_{\max}$, $P_{\mathrm{W}}^{(c)}(P_{\mathrm{i}}, t) \equiv 1$.

Finally, for $\tilde{o}_{\mathrm{c}} = 1$, the solution can be written as $P_{\mathrm{W}}^{(c)}(P_{\mathrm{i}}, t) = 1 - e^{-B_{\mathrm{W}}^{(c)}(P_{\mathrm{i}})t}$.

The above discussion allows us to write the time dependent expression of $\mathrm{NA_W(P4)}$. From the result it is obvious that the availability degrades with time. Its upper bound, corresponding to $P_{\mathrm{W}}^{(c)} = 1$, and that will be reached if $P_{\mathrm{W}}^{(c)}$ grows fast (or if the partition lasts long enough), is given by

$$\begin{aligned}\lim_{t \gg} \mathrm{NA_W(P4)} = \sum_{i=1}^{q} \frac{P_{\mathrm{i}}}{N} \Bigg\{ & \left[1 - \left(1 - \left(1 - \frac{r}{N}\right)^{P_{\mathrm{i}}}\right)^{\tilde{o}_{\mathrm{I}}}\right] + \\ & \left[1 - \left(1 - \frac{r}{N}\right)^{P_{\mathrm{i}}}\right]^{\tilde{o}_{\mathrm{I}}} \\ & \left[1 - \left(1 - \frac{o_{\mathrm{c}} R_{\mathrm{c}}}{M}\right)^{o_{\mathrm{D}}}\right] \Bigg\}\end{aligned} \tag{9}$$

In this limit, no writing invocation is allowed if it involves any critical constraint.

Reading invocations. The non-availability of reading operations in this model is exactly the same as in the case of *primary partition*, since the only source of failure for a reading operation is a missing replica for some of the directly accessed objects.

$$\mathrm{NA_R(P4)} = \mathrm{NA_R(PP)}. \tag{10}$$

6 Evaluation

The model introduced in the previous section has been used to evaluate the P4 protocol against the primary partition model. To this end, some of the parameters identified in section 5.2 have been fixed to certain values. The values used have been taken from a real-world distributed object system. The Distributed Telecommunication Management System (DTMS) is one of the target applications of DeDiSys. The system monitors and controls a distributed voice communication system used in air traffic control.

One hundred nodes with a total of 500 objects and 10 replicas per object are assumed. An average of 3 objects are involved in both critical and regular constraints. On average, 10 invocations are launched per time unit. Furthermore, as partitioning is assumed to take a while to repair, all the availabilities have been calculated after one hour of partitioning. The reason for this one hour delay was to take into account the degradation of availability that can occur in the P4 protocol due to overlapping constraints. That is, in the P4 protocol an operation cannot proceed if an associated constraint contains a revocable object.

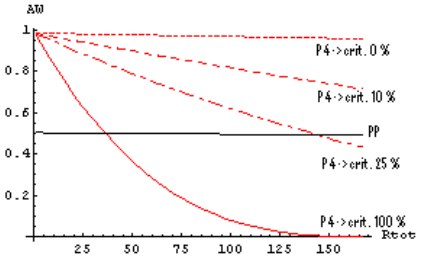


Fig. 1. Availability vs. number of constraints

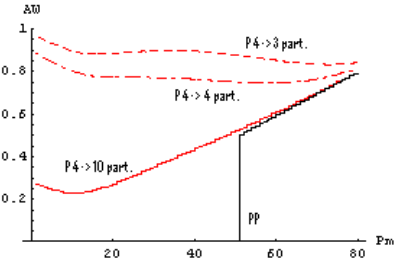


Fig. 2. Availability vs. partition size

Figure 1 shows the availability of write operations that was calculated for the two models plotted against the total number of constraints in the system. The percentage of critical constraints was varied. The availability of the primary partition model remains almost constant at about 0.5, since partitions of almost the same size are assumed.

In the case of the P4 protocol, firstly, the ideal case of 0% of critical constraints has been plotted to show the best case availability. As can be seen, availability remains close to the maximum of 1 and decreases very slowly. Secondly, the cases of 10% and 25% of critical constraints have been plotted to demonstrate a realistic application scenario. In DTMS the ratio of critical constraints is approximately 10%. As can be seen, in these cases availability decreases relatively slowly. In the DTMS case a very high number of constraints would be needed for the primary partition model to provide better availability. Finally, the worst case scenario is demonstrated by setting the ratio of critical constraints to 100%. Availability, in this case, decreases relatively fast and the primary partition model starts to perform better at a relatively low number of constraints.

As can be seen the P4 performs well, if the number of critical constraints is relatively low. The P4 protocol should be used in applications that are able to tolerate temporary relaxation of consistency, whereas the primary partition model performs better for applications which require a high level of consistency. In the DTMS scenario the P4 protocol provides the overall better availability.

As a further analysis, availability of the two models has been plotted against the number of nodes in a partition in the system. (Figure 2). 0% critical constraints were assumed in this case. The number of partitions in the system has

been plotted at 3, 4 and 10. The horizontal axis lists the number of nodes in the partition as a percentage of the total number of nodes in the system. As can be seen, availability of the primary partition model is 0 until a majority partition exists. This is because in the primary partition model no partition can continue, if there is no majority.

In contrast the P4 protocol provides availability, even when no majority exists. Furthermore, availability of the P4 is higher than that of the primary partition model for all number of nodes in one partition in this case (no critical constraints). The availability of the P4 model is higher, if there are fewer partitions. This is due to the probability that a replica is available in the local partition.

7 Conclusion and Future Work

We have designed a replication protocol for increased availability in replicated distributed object systems. The protocol allows consistency to be temporarily relaxed, in order to increase the availability. Consistency is based on integrity constraints.

In a partitioned system we allow operations to continue in each partition, in contrast to the primary partition approach, in which only a majority partition may proceed. We have developed an analytical model of both approaches, in order to compare them. The results indicate that the new protocol increases the availability significantly for applications with a low number of critical integrity constraints; that is, for applications where consistency can be relaxed. Furthermore, the protocol has the advantage that operations can proceed, even if the accessed object is not in a majority partition.

We are currently implementing the P4 protocol in the CORBA version of the DeDiSys middleware [19], in order to verify the results obtained by the analytical study presented in this paper and the simulation results obtained in our prototype environment [3]. Furthermore, we plan to investigate further replication strategies tailored for different target applications of DeDiSys.

References

1. Ricciardi, A., Schiper, A., Birman, K.: Understanding partitions and the "non partition" assumption. In: IEEE Proc Fourth Workshop on Future Trends of Distributed Systems. (1993)
2. Osrael, J., Froihofer, L., Goeschka, K.M., Beyer, S., Muñoz-Escoí, F.D., Galdámez, P.: A system architecture for enhanced availability of tightly coupled distribute d systems. In: International Conference on Availability, Reliability and Security. (2006) 400–407
3. Beyer, S., Sánchez, A., Muñoz-Escoí, F.D., Galdámez, P.: Dedisys lite: An environment for evaluating replication protocols in partitionable distributed object systems. In: International Conference on Availability, Reliability and Security. (2006) 408–415
4. Demers A. J. et al.: The bayou architecture: Support for data sharing among mobile users. In: Proceedings IEEE Workshop on Mobile Computing Systems & Applications. (1994) 2–7

5. Gray, J., Helland, P., O'Neil, P., Shasha, D.: The dangers of replication and a solution. In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. (1996) 173–182
6. Preguiça N. et al.: Mobile transaction management in mobisnap. In: Current Issues in Databases and Information Systems: East-European Conference on Advances in Databases and Information Systems, Springer LNCS (2000) 379–386
7. Shapiro, M., Rowstron, A., Kermarrec, A.M.: Application-independent reconciliation for nomadic applications. In: EW 9: Proceedings of the 9th workshop on ACM SIGOPS European workshop. (2000) 1–6
8. Phatak, S.H., Badrinath, B.R.: Multiversion reconciliation for mobile databases. In: ICDE '99: Proceedings of the 15th International Conference on Data Engineering, IEEE Computer Society (1999) 582
9. Kozlova, A., Kochnev, D., Novikov, B.: Efficient consistency support for distributed mobile applications. In: Proceedings of the Spring Young Researcher Colloquium on Database and Information Systems. (2004) 31–41
10. Yu, H., Vahdat, A.: Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans. Comput. Syst. **20** (2002) 239–282
11. Ferdean, C., Makpangou, M.: A generic and flexible model for replica consistency management. In: ICDCIT. (2004) 204–209
12. Cukier M. et al.: Aqua: An adaptive architecture that provides dependable distributed objects. In: SRDS '98: Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems. (1998) 245
13. Bañuls, M., Galdámez, P.: Extended membership problem for open groups: Specification and solution. In: VECPAR 2004: High Performance Computing for Computational Science. Number 3402 in LNCS (2004) 288–301
14. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In: Distributed systems. 2nd edn. ACM Press, Addison-Wesley (1993) 97–145
15. Cristian, F.: Understanding fault-tolerant distributed systems. Commun. ACM **34** (1991) 56–78
16. Schneider, F.B.: What good are models and what models are good? In: Distributed Systems. 2nd edn. ACM Press, Addison-Wesley (1993) 17–26
17. Budhiraja, N., Marzullo, K., Schneider, F.B., Toueg, S. In: The primary-backup approach. ACM Press, Addison-Wesley (1993) 199–216
18. Guerraoui, R., Schiper, A.: Software-based replication for fault tolerance. Computer **30** (1997) 68–74
19. Beyer, S., Muñoz-Escoí, F.D., Galdámez, P.: Corba replication support for fault-tolerance in a partitionable distributed system. In: Second International Workshop on High Availability of Distributed Systems. (2006)

Exploiting Multidomain Non Routable Networks

Franco Frattolillo and Salvatore D'Onofrio

Research Centre on Software Technology
Department of Engineering, University of Sannio, Italy
frattolillo@unisannio.it

Abstract. Exploiting computing resources available within "departmental" organizations to run large-scale applications can be considered a difficult task since such resources are usually represented by computing nodes that belong to non-routable, private networks and are connected to the Internet through publicly addressable IP front-end nodes. This paper presents a Java middleware that can support the execution of large-scale, object-based applications over heterogeneous multidomain, non-routable networks. Furthermore, the middleware can be exploited to relieve programmers of the classic burden tied to the deployment of PVM runtime libraries and program executables among computational resources belonging to distinct administrative domains.

1 Introduction

Middlewares make it possible to exploit the enormous, but often poorly utilized, computing resources existing on the Internet in order to solve large-scale problems. They extend the concept, originally introduced by PVM [1], of a "parallel virtual machine" restricted and controlled by a single user, thus enabling the building of computing systems, called "metacomputers", composed of heterogeneous computing resources of widely varying capabilities, connected by potentially unreliable, heterogeneous networks and located in different administrative domains [2]. However, the peculiarities characterizing such a computational context require that middlewares deal with highly variable communication delays, security threats, machine and network failures, and the distributed ownership of computing resources in order to support programmers in configuring and optimizing their large-scale applications. Therefore, middlewares can build on most of the current distributed and parallel software technologies, but they require further advances in techniques and tools to bring together computational resources distributed over the Internet to efficiently solve a single large-scale problem according to the scenario introduced by grid computing [2].

On the other hand, cluster computing still remains a valid alternative to grid computing, since clusters of workstations represent high performance/cost ratio computing platforms, and are widely available within the so called "departmental" organizations, such as research centres, universities, and business enterprises [3]. However, workstation clusters are usually exploited by running MPI or PVM applications within trusted and localized network environments,

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 696–709, 2006.
© Springer-Verlag Berlin Heidelberg 2006

where problems concerning security, ownership, and configuration of the used networked resources are easily solved within the same administrative domain. Furthermore, it is also worth noting that most of the computing power existing within departmental organizations is often represented by computing nodes that belong to non-routable, private networks and are connected to the Internet through publicly addressable IP front-end nodes [3]. Therefore, it cannot be considered actually available to run large-scale applications, since it cannot be easily exploited by many currently used middlewares for grid computing [2,3].

The considerations reported above suggest that a middleware should be able to use computing resources across multidomain, non-routable networks and relieve programmers of the usual burden of manually deploying PVM or MPI runtime libraries and program executables among computational resources belonging to distinct administrative domains [4]. Furthermore, the need for adaptability requires that a middleware is also characterized by a flexible implementation developed according to a component-based, reflective architecture [5] in order to facilitate dynamic changes in the configuration of the built metacomputers.

Java has been widely used to develop middlewares for metacomputing. It has been designed for programming in heterogeneous computing environments, and provides a direct support to multithreading, code mobility and security, thus facilitating the development of concurrent and distributed applications. However, most of the Java middlewares do not often adequately support the programming and execution of dynamic parallel applications on multidomain, non-routable networks. Furthermore, they often exploit tightly-coupled interaction and communication paradigms based on the "message-passing" model or on the Java RMI package, thus lacking specific coordination mechanisms able to manage the resource variability in the configuration of metacomputers.

This paper presents a customizable middleware that can support the execution of large-scale, object-based applications over heterogeneous multidomain, non-routable networks. The middleware, called Java Multidomain Middleware ($JMdM$), can exploit computing resources hidden from the Internet, but connected to it through publicly addressable IP front-end machines, as computing nodes of a unique metacomputer. Thus, the computing power existing within departmental organizations as non-publicly IP addressable computing nodes can be made available to run applications without having to exploit low-level, "ad hoc" software libraries or specific systems or resource managers for grid computing, which could turn the development of parallel applications into a burdensome activity as well as penalize application performance [4,6]. Finally, $JMdM$ has also been developed to support programmers in deploying ePVM [3] applications among computational resources belonging to multidomain clusters.

The outline of the paper is as follows. Section 2 presents $JMdM$ and describes the architecture of the metacomputers that can be built by using the middleware. Section 3 describes the main implementation details of $JMdM$. Section 4 describes how ePVM applications can be deployed by exploiting $JMdM$. Section 5 reports on some experimental results. Finally, in Section 6 conclusion remarks are available.

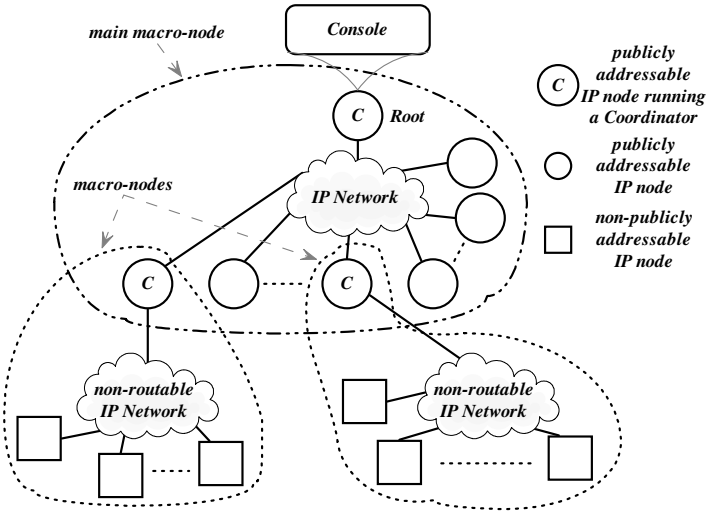


Fig. 1. The architecture of a metacomputer

2 The Proposed Middleware

JMdM is a Java-based middleware designed according to the reference model originally described in [7]. It makes it possible to build and dynamically reconfigure a metacomputer on which large-scale applications can be deployed and run. The configured metacomputer can aggregate computing resources directly available on the Internet as well as those belonging to multidomain, non-routable networks, i.e. computing nodes not provided with public IP addresses, but connected to the Internet through publicly addressable IP front-end nodes. Thus, even though the computing nodes composing the metacomputer result in being arranged according to a hierarchical physical network topology consisting of two levels (the level of the publicly addressable IP nodes and the level of the not publicly addressable IP nodes belonging to non-routable networks), they virtually appear as arranged according to a flat network topology (see Figure 1).

Computing nodes can be PCs, workstations or computing units of parallel systems interconnected by heterogeneous or dedicated networks. However, *JMdM* enables programmers to exploit the features of the underlying physical computational and network resources in a transparent way. To this end, the metacomputer is viewed as abstractly composed of computational *node*s interconnected by a flat virtual network and unambiguously identified by integer values assigned at the metacomputer start-up. More precisely, the *node*s hidden from the Internet or connected by dedicated networks can be grouped in *macro-node*s, which thus abstractly appear as single, more powerful, virtual computing units. The metacomputer is assumed to be made up by at least one *macro-node*, called the *main macro-node*, which groups all the publicly addressable IP computing *node*s taking part in the metacomputer.

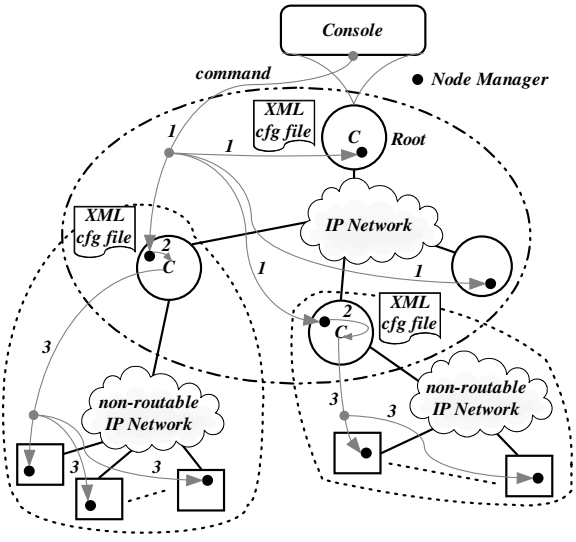


Fig. 2. The scheme of a command execution

Each *node* maintains status information about the run-time architecture of the metacomputer, such as information about the identity and liveness of the other *node*s. In fact, the hierarchical physical organization of the metacomputer allows each *node* to keep and update only information about the configuration of the *macro-node* which it belongs to, thus promoting scalability.

Each *macro-node* is managed by a special *node* called *Coordinator* (C) that:

- is allocated onto the publicly addressable IP *node* of each non-routable network interconnecting other non-directly addressable IP *node*s;
- creates the *macro-node* by activating *node*s within the private network that it manages;
- takes charge of updating the status information of each *node* grouped by the *macro-node*;
- monitors the liveness of *node*s to dynamically change the configuration of the *macro-node*;
- carries out the automatic "garbage collection" of the crashed *node*s in the *macro-node*;
- acts as a "gateway", since it
  - forwards system communications to the internal *node*s of the *macro-node*;
  - enables *node*s belonging to distinct *macro-node*s of the metacomputer to directly communicate.

The metacomputer is controlled by the *Coordinator* of the *main macro-node*, called *Root*, which is directly interfaced with the user through the *Console*, by which the metacomputer configuration can be dynamically managed (see Figure 1). To this end, each *node* wanting to make its computing power available to *JMdM* runs a special server, called *Node Manager* (NM), which takes

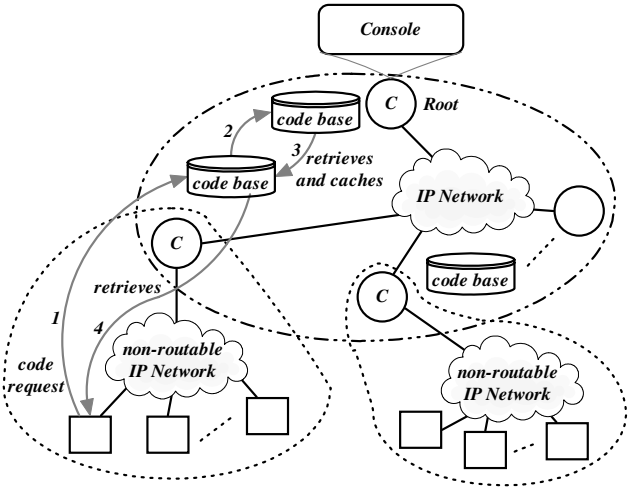


Fig. 3. The Distributed Class Storage System

charge of interacting with the *Console* as well as running further software components that enable the *node* to participate in the metacomputer.

The *Console* can only interact with the NMs running on publicly addressable IP *node*s, i.e. the *node*s belonging to the *main macro-node*. Therefore, when a NM hosted by a *node* running a *Coordinator* receives a command from the *Console*, such as, for example, a "configuration" command, it has to forward it to the NMs of the *node*s grouped by its *macro-node*. Then, the NM contacts the *Coordinator* of its *macro-node*, which exploits the configuration information stored in a specific XML file to forward the received command to the NMs of the *node*s inside the *macro-node*. In fact, information contained in the XML file is initially provided by the administrator of the *node*s grouped by the *macro-node*, and then dynamically updated by the *Coordinator* (see Figure 2).

All the *node*s can exchange messages according to a communication semantics based on the "one-sided" model [8]. Therefore, each *node* can directly send objects, which can be either simple messages or tasks. Objects can be sent also to the *node*s that do not store the object code. To this end, each *node* is provided with a "code loader", which can retrieve the object code from a distributed code repository, called *Distributed Class Storage System* (DCSS), purposely developed to ensure scalability to the proposed middleware (see Figure 3).

The architecture of the DCSS follows the global architecture of the metacomputer. Therefore, each *macro-node* is provided with a dedicated "code base" managed by the *Coordinator*. When a *node* needs a code, it requires the *Coordinator* of its *macro-node* to retrieve it. If the *Coordinator* lacks the required code, it asks the *Root* for it, which stores all the code of the running application. However, whenever a *Coordinator* obtains the required code, it stores it in a local cache, in order to reduce the loading time of successive code requests coming from other *node*s of the *macro-node*.

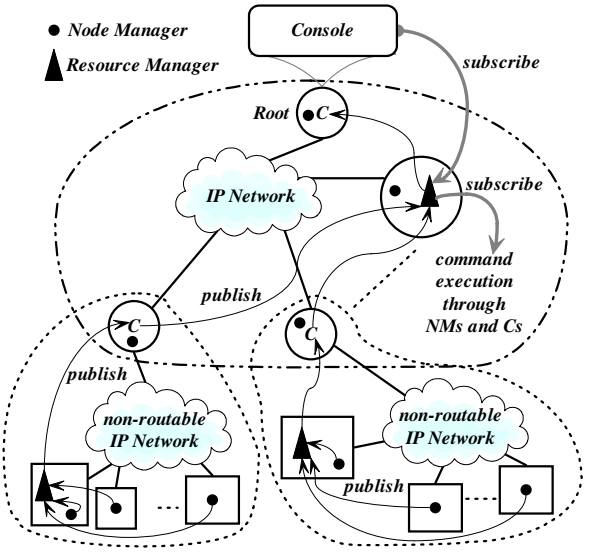


Fig. 4. The publish/subscribe information service

JMdM also provides a "publish/subscribe" information service implemented by two servers: the *Resource Manager* (RM) and the NM (see Figure 4). In particular, each *macro-node* has an RM, which can be allocated onto one of the *node*s belonging to the *macro-node*. A RM is periodically contacted by the NMs of the *node*s belonging to the *macro-node* and wanting to publish information about the CPU power and its utilization or the available memory or the communication performance. Information is collected by the RM and made then available to the "subscribers", which are the *Coordinator*s belonging to the metacomputer. Thus, each *Coordinator* can know the maximum computing/communication power made available by its *macro-node*. Furthermore, this information is also made available to the *Root*, which can thus know the power of all the *macro-node*s making up the metacomputer. This allows users to know the globally available computing power and reserve a part of it by issuing a "subscription" request to the *Console*. Then, the *Console* can ask the RM of the *main macro-node* for selecting and reserving only the required computing resources. The result of this process is an XML file containing system information to create a metacomputer without having to consult anew the RM.

3 The Implementation of the Middleware

JMdM has been designed according to a component-based, reflective approach that enables the middleware's software architecture to dynamically adapt to changes in the configuration of physical components [5]. Therefore, all the services supplied by *JMdM* are implemented by software components whose

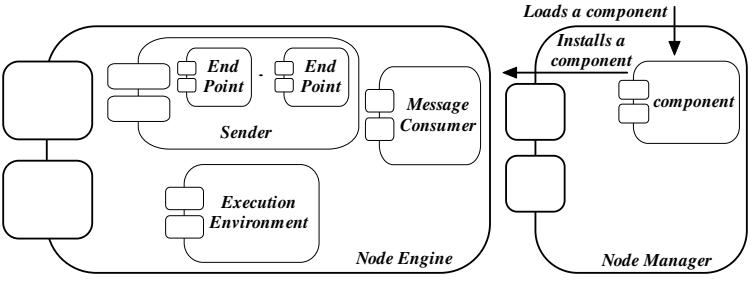


Fig. 5. The main components of a *node*

interfaces define an abstract design able to provide solutions for metacomputing problems. Such a design is concretized by the definition of classes, which implement the interfaces of the abstract design and interact with the basic classes representing the skeleton of the middleware architecture. This approach gives the possibility of modifying or substituting a component implementation without affecting other parts of the middleware, and allows a component to have different implementations, each of which can be dynamically loaded in the middleware.

Based on the design principles reported above, *Coordinator*s and *node*s are all implemented as processes that can load a set of software components either at start-up or at run-time. In particular, the main components loaded by each *node* are (see Figure 5): the *Node Manager* (NM) and the *Node Engine* (NE).

The main tasks of the NM consist in storing system information and interacting with the *Coordinator* in order to guarantee the macro-node consistency. The NE takes charge of receiving application messages from the network and processing them. In fact, the NM implements some system tasks, and so it:

- monitors the liveness within the metacomputer by periodically sending control messages to the *Coordinator* of its *macro-node*;
- kills the *node* when the macro-node *Coordinator* is considered crashed;
- creates the NE by using the configuration information supplied by the *Coordinator* of its *macro-node*.

The NE implements the node behavior by installing a set of components, which are dynamically loaded by a specific component of the NM, called the *Loader* (LD). In fact, the components loaded by the NE are all configurable, and make it possible to customize the node behavior in order to provide applications with the necessary programming or run-time support. However, applications are not allowed to directly access the NE, but they can exploit its features or change its behavior through the NM. As a consequence, the NM also represents the interface between the application and the services provided by the middleware, such as the metacomputer reconfiguration and management.

The configurable components of the NE are: the *Execution Environment* (EE), the *Sender* (SD) and the *Message Consumer* (MC) (see Figure 5). They implement

the abstractions that define the execution behavior of a *node*: a passed on to the MC, which is the interface between an incoming communication channel and the EE. Then, the message is delivered to the EE, which processes it according to the strategy coded in the EE implementation. As a result of message processing, new messages can be created and sent to other *node*s by using the SD.

The EE defines the node behavior. It can contain either application components or data structures necessary to run parallel or distributed applications according to a specific programming model. Therefore, since each *node* can handle a different EE implementation, MIMD applications can be run by simply distributing their components wrapped in the implementations of the EE of each *node*. Furthermore, EE can dynamically control the configuration of the metacomputer as well as exploit the services implemented by the other node components, since it has access to the NM. Thus, a running application can both evolve on the basis of the configuration information characterizing the metacomputer and dynamically configure it, by adding, for example, a *node*, if more computing power is necessary.

The SD implements services for routing the messages generated on a *node* towards all the other *node*s of the metacomputer. In particular, $JMdM$ provides a default implementation for this component, called *Default Sender* (DS), which is loaded if any other SD is not installed by the user application.

The DS implements basic communication mechanisms that can exploit the multidomain network organization of the metacomputer and support the development of more sophisticated communication primitives, such as the ones based on synchronous or collective messaging. It exploits some components of the NE not accessible to the user, called *End Points* (EPs).

An EP is the local image of a remote *node* belonging to the *macro-node*. It manages a link to the transport module selected during the configuration phase of the metacomputer to enable communications, through a specific protocol, towards the remote *node* represented by the EP. To this end, the transport modules are characterized by a communication interface independent of the underlying transport protocol used to manage communications. Thus, a transport module implementing a new communication protocol or exploiting a native communication library can be easily integrated in $JMdM$ by only developing a specific module, called *adapter*, able to abstract from the implementation details characterizing the low-level communications. In fact, every *adapter* has to carry out the serialization of the objects sent by the application according to the specific features implemented by the corresponding transport module. In particular, $JMdM$ implements, by default, three *adapter*s. The first is based on TCP, the second extends UDP by adding reliability, and the third is based on "Fast Messages" [9] for Myrinet networks.

An MC is a component whose reference is passed on to all the transport modules installed on the *node*. It implements the actions that have to be performed on the *node*, according to the strategy of message consuming defined by the programmer, whenever a message is received from the network.

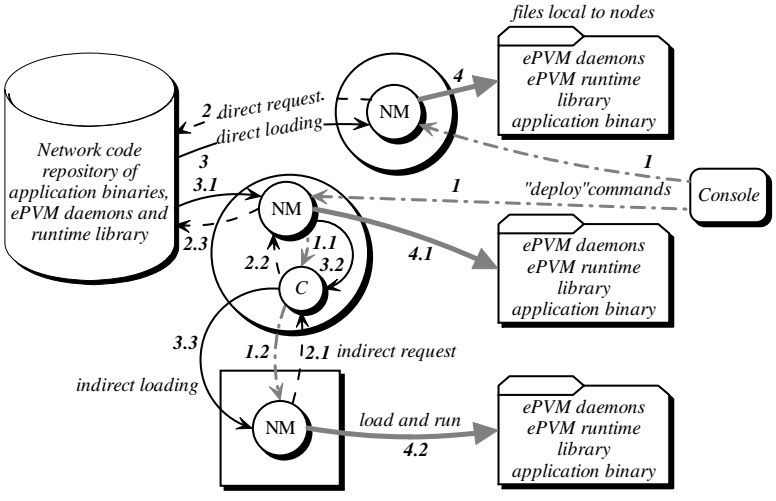


Fig. 6. The deployment of an ePVM application

4 Deploying ePVM Applications

As described in the previous section, *JMdM* implements an architecture that can host dynamically loaded software components. To this end, it supplies a Java runtime environment in the form of a component container. In fact, components can be uploaded by users who have appropriate permissions, while the administrators of the *node*s aggregated by *JMdM* retain complete and fine-grained control over the computing and network resources they manage. In particular, each NM may request the Java Virtual Machine (JVM) running on each *node* to link a dynamic native library. More precisely, a NM can transparently resolve and link at runtime the appropriate version of a library precompiled for different platforms by following a four-steps scheme. Therefore, the NM:

1. obtains the library path referred to the code repository;
2. resolves the actual Java "resource" name by combining information about the library path, requested library name, and detected platform type;
3. stages the resource to a temporary local file;
4. loads the library from that file.

To this end, NMs assume an automatic platform detection type based on the classification method adopted by PVM [1]. Furthermore, *JMdM* assumes that the loaded resources may originate from an arbitrary URL.

ePVM [3] is an extension of PVM developed to enable PVM applications to run across multidomain clusters made up by computing nodes belonging to non-routable, private networks, but connected to the Internet through publicly addressable IP front-end nodes. In particular, to run an ePVM application, the only files that have to be present at the *node*s aggregated by *JMdM* are: the

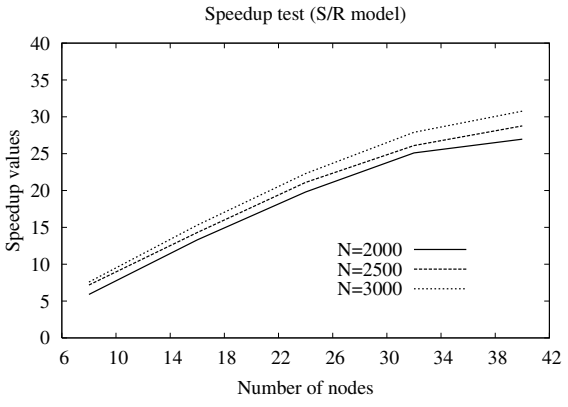


Fig. 7. Speedup values obtained by using all publicly addressable IP nodes

ePVM daemons, the ePVM runtime library, and the application itself [3] (see Figure 6). In fact, ePVM has not external dependencies except for the standard C library, and this makes precompiled versions of these files portable within a single platform type. Therefore, to set up the ePVM runtime, the *Console* requests NMs to load the ePVM daemons and runtime library. Then, NMs take charge of staging and starting up the appropriate platform-specific versions of the ePVM daemons, fetching them from a network code repository. To this end, it is worth noting that the execution of the "set up" command requested by the *Console* is managed by $JMdM$ as a normal command, according to what shown in Figure 2. This means that only the NMs belonging to the publicly addressable IP *node*s can directly load the ePVM runtime, whereas the NMs running on *node*s inside *macro-node*s can load the runtime through their *Coordinator*s.

It is worth noting that *node* administrators have only to authorize the deployment, but they have not to configure the ePVM runtime, since NMs can stage and start up the appropriate platform-specific versions of the ePVM daemons and runtime library. Furthermore, users are allowed to execute ePVM applications on computing nodes where they have not a login account but instead, only restricted $JMdM$ access. As a consequence, the setup procedure implemented by $JMdM$ hides heterogeneity from users as well as releases *node* administrators from the responsibility to install or configure additional software except for the $JMdM$ support itself.

The ePVM runtime setup is directly followed by the application deployment (see Figure 6). To this end, the application binary has to be stored in a specific directory, designated by the ePVM daemons, of the file system of each *node*. Therefore, the *Console* can ask NMs for loading the application binary from a URL code base that the user may specify along with the executable name. As illustrated in Figure 6, the execution request coming from the *Console* is handled by publicly IP addressable NMs, which use their already described capabilities to fetch the platform-specific versions of the application and save them in files local to *node*s. Then, these NMs ask their *Coordinator*s for forwarding the execution

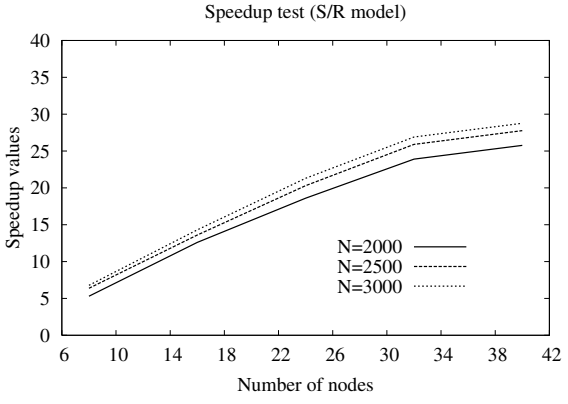


Fig. 8. Speedup values obtained by exploiting three PC clusters configured as non-routable, private networks

request to NMs running on the *node*s inside *macro-node*s, thus enabling the deployment of the application on all the *node*s aggregated by the metacomputer.

After the deployment of the application, each NM can directly invoke the ePVM daemons to launch the application from the local file. However, to provide protection against malicious code, NMs can assess whether the deployed application can be trusted. To this end, NMs can base their decisions upon the code source and/or user who requested the execution, according to the basic Java security model. In particular, NMs can restrict the code source to designated places that can be specified as URL base paths and/or verify code signatures and/or use flexible authentication mechanisms to determine user identity.

5 Experimental Results

This Section reports on some performance experiments conducted by exploiting three different clusters of PCs connected by a Fast Ethernet network.

The first cluster is composed of 16 PCs connected by a Fast Ethernet hub and equipped with Intel Pentium IV 3 GHz, hard disk EIDE 60 GB, and 1 GB of RAM. The second cluster is composed of 16 PCs connected by a Fast Ethernet switch and equipped with Intel Xeon 2.8 GHz, hard disk EIDE 80 GB, and 2 GB of RAM. The third cluster is composed of 8 PCs interconnected by a Fast Ethernet hub and equipped with Intel Pentium IV 2.5 GHz, hard disk EIDE 40 GB, and 1 GB of RAM. All the PCs use the release 5.0 of the SUN JDK.

Two classes of experiments have been conducted. In the former, all the PCs belonging to the clusters have been provided with public IP addresses, and this has made it possible to build a metacomputer characterized solely by the *main macro-node*. In the latter, the PCs belonging to the clusters have been configured so as to form three private, non-routable networks. Therefore, the built metacomputer has been composed of three *macro-node*s.

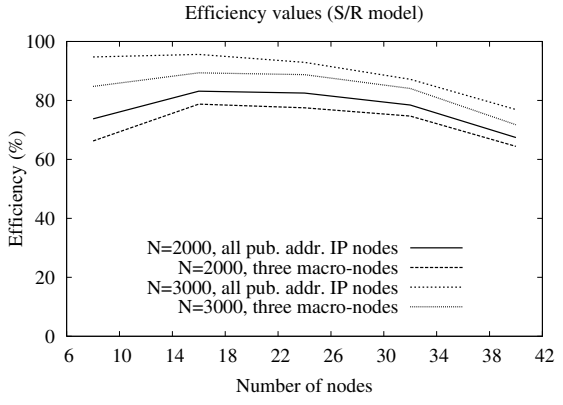


Fig. 9. Comparison of the efficiency values shown in the Figures 7 and 8

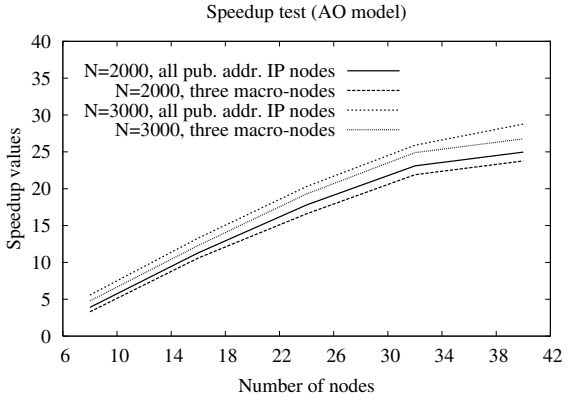


Fig. 10. Speedup values obtained by exploiting the AO programming model

In both the classes of experiments, the parallel product of two square matrices, whose size is N, has been executed. The product has been implemented by using the "striped partitioning": the right matrix is transferred and allocated on each *node* of the metacomputer, whereas the left matrix is split in groups of rows each one transferred to a different *node*. In particular, the speedup values have been measured under a varying number of *node*s building the metacomputer. The *node*s have been evenly allocated among the PCs of the three clusters.

The product has been programmed according to two different programming models: the "Send/Receive" (S/R) model and the "Active Objects" (AO) model. The former is the well-known message-passing model used to program parallel and distributed applications, whereas the latter is the model proposed in [10,11] to overcome some limitations of the classic message-passing and RPC models.

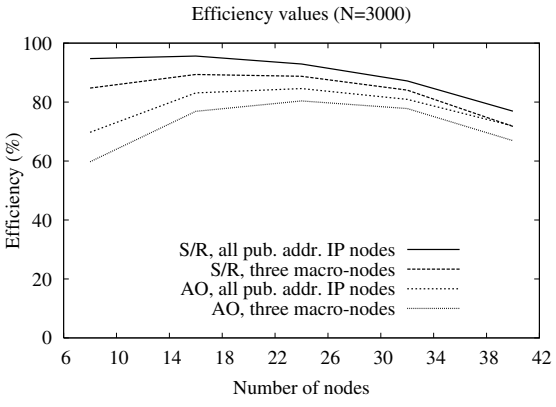


Fig. 11. Comparison of the efficiency values achieved by exploiting both the S/R and the AO model

Both models have been implemented by customizing the NE components, according to what reported in the previous sections. Moreover, the TCP transport protocol has been used among all the *node*s of the metacomputer.

Figure 7 and Figure 8 show the speedup values obtained by running the product of matrices programmed by exploiting the S/R model. The curves show that the speedup values tend to increase with the number of *node*s depending on the size of the matrices. In particular, the PC clusters exploited in the test of Figure 8 have been interconnected by a Fast Ethernet network and have been interfaced to a further multi-homed PC used to only run the *Console*.

Figure 8 shows that the speedup values obtained with the multidomain network configuration are similar to the ones shown in Figure 7. In fact, Figure 9 shows that the performance penalization determined by the overhead caused by the management of the *macro-node*s is rather limited and within 10%.

The results obtained by employing the S/R programming model are essentially confirmed by exploiting the AO programming model, as shown by Figure 10. In fact, Figure 11 shows that the efficiency values achieved in the two classes of experiments are rather similar, with differences within 10%. This demonstrates that $JMdM$ is characterized by a good flexibility that is achieved without reducing the performance of the executed applications.

6 Conclusions

In this paper a customizable and reflective middleware able to run parallel and distributed object-based applications across heterogeneous multidomain, non-routable networks is presented. The middleware enables all the computing power available within departmental organizations and non-directly IP addressable to be harnessed to run applications without having to exploit low-level, "ad hoc" software libraries or specific systems or resource managers for grid computing,

which could turn the development of parallel applications into a burdensome activity as well as penalize application performance.

The middleware enables the building of metacomputers made up by abstract computing *node*s, each of which can dynamically install interacting components that can be purposely customized in order to adapt the programming model supported by the metacomputer to the application needs. Furthermore, the middleware also supports programmers in deploying ePVM applications among computational resources belonging to multidomain clusters.

Finally, the proposed middleware has been tested by conducting some experimental tests programmed by implementing two different programming models. In fact, the obtained results demonstrate that flexibility can be achieved without reducing performance.

References

1. Geist, A., Beguelin, A., , et al.: PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing. The MIT Press (1994)
2. Foster, I., Kesselman, C.: The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, San Mateo, California, USA (2004)
3. Frattolillo, F.: Running large-scale applications on cluster grids. Int'l Journal of High Performance Computing Applications **19**(2) (2005) 157–172
4. Kurzyniec, D., Hwang, P.and Sunderam, V.: Failure resilient heterogeneous parallel computing across multidomain clusters. Int'l Journal of High Performance Computing Applications **19**(2) (2005) 143–155
5. Parlavantzas, N., Coulson, G., et al.: Towards a reflective component based middleware architecture. In: Procs of the Int'l Workshop on Reflection and Metalevel Architectures, Sophia Antipolis and Cannes, Australia (2000)
6. Schopf, J.M., Nitzberg, B.: Grids: The top 10 questions. Scientific Programming, special issue on Grid Computing **10**(2) (2002) 103–111
7. Foster, I., Kesselman, C., Tuecke, S.: The anatomy of the grid: Enabling scalable virtual organizations. Int'l Journal of Supercomputer Applications **15**(3) (2001) 200–222
8. von Eicken, T., Culler, D.E., et al.: Active messages: A mechanism for integrated communication and computation. In: Procs of the 19th ACM Int'l Symposium on Computer Architecture, Gold Coast, Queensland, Australia (1992) 256–266
9. Lauria, M., Pakin, S., Chien, A.A.: Efficient layering for high speed communication: Fast messages 2.x. In: Procs of the 7th High Performance Distributed Computing Conference, Chicago, Illinois, USA (1998)
10. Di Santo, M., Frattolillo, F., et al.: An approach to asynchronous object-oriented parallel and distributed computing on wide-area systems. In: Procs of the Int'l Workshop on Java for Parallel and Distributed Computing. Volume 1800 of LNCS., Cancun, Mexico (2000) 536–543
11. Di Santo, M., Frattolillo, F., et al.: A component-based approach to build a portable and flexible middleware for metacomputing. Parallel Computing **28**(12) (2002) 1789–1810

Recovery Strategies for Linear Replication*

Rubén de Juan-Marín, Luis Irún-Briz, and Francesc D. Muñoz-Escoí

Instituto Tecnológico de Informática
Universidad Politécnica de Valencia
Camino de Vera, s/n - 46022 Valencia, Spain
{rjuan, lirun, fmunyoz}@iti.upv.es

Abstract. Replicated systems are commonly used to provide highly available applications. In last years, these systems have been mostly based on the use of atomic broadcast protocols, and a wide range of solutions have been published. The use of these atomic broadcast-based protocols also has aided to develop recovery protocols providing fault tolerance to replicated systems. However, this research has been traditionally oriented to replication systems based on constant interaction for ensuring 1-copy-serializability. This paper presents a general strategy for recovery protocols based on linear interaction as well as providing other isolation levels as snapshot isolation. Moreover, some conclusions of this work can be used to review recovery protocols based on constant interaction.

1 Introduction

Fault tolerance, high availability and performance are success keys in nowadays information systems. Consequently, distributed systems have been widely expanded among organizations and enterprises in order to provide these characteristics.

Particularly, replicated systems are the most common way used to reach these goals, being replicated databases an example of typical application. Therefore, several replication techniques have been largely studied and a wide range of proposals have been implemented. Latest trends in replication techniques are oriented to make use of group communication system (GCS) semantics. In fact, most non-commercial solutions combine eager update propagation with constant interaction [1] using atomic broadcast protocols [2], providing more efficient implementations. A wide number of approaches [3], [4], [5] are described in the literature.

Applications built on top of GCS (e.g. replication protocols) make use of GCS membership mechanisms which manage group partitions [6]. On one hand, this mechanism excludes disconnected, failed or partitioned nodes from the group, notifying the changes to survivor nodes. On the other hand, it allows new incorporations to the group or node reconnection, also notifying the membership changes to the group members.

Usually, these membership changes may originate outdated nodes, i.e. replicas that have lost some updates, and therefore without the last state. But traditionally, replication protocols do not give great relevance to the outdated nodes recovery, not being a real fully-functional fault tolerant system. So it is needed a new architecture component

* This work has been partially supported by the Spanish MCYT grant TIC2003-09420-C02-01.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 710–723, 2006.
© Springer-Verlag Berlin Heidelberg 2006

that knows what to do when: a node failure occurs, a failed node rejoins to the group or a new node is added to the replicated system. The most important function of this component consists of updating those replicas with an outdated system state before they become full-functional system nodes. Therefore, the recovery protocol must include in the replication protocol additional actions, in order to store and maintain the information used in the recovery processes (whenever it is needed).

The outdated nodes recovery can be done in many ways, ranging from the simplest one (backup transfers) to more complex alternatives. Ideally, this process must be performed without disrupting the replicated system work. In this direction, a wide variety of recovery protocols has been presented in the literature scoped on replicated databases, as [3], [7], [8] most of them based on group communication.

This work presents a general strategy for recovery protocols based on linear interaction, in contrast of using the constant interaction [1] approach. Linear interaction, in spite of its high performance cost, will be the only feasible alternative for object-oriented replicated systems with large data states to transfer, and with a transactional support, such as FT-CORBA with its complementary *Transaction Service*, where constant interaction will either lead to huge messages or be impractical in case of partial replication, since the state to be transferred should be collected from different source nodes. But, as it will be shown, the management required by linear-recovery protocols is more complex because it must manage multiple messages per transaction. In addition, for ensuring correctness under linear interaction, messages belonging to not-yet-committed (as well as for rolled-back) transactions, must be adequately treated.

In parallel, the proposed recovery strategy adopts the *crash-recovery with partial-amnesia* failure model because it supports the recovery of outdated nodes. Recovery which becomes a key point for building fully-functional fault tolerant systems in replicated systems with large data states. The traditional adopted failure model, *crash* or *fail-stop*, is not adopted because does not support outdated nodes recovery presenting only good behavior for replicated systems with few data state.

The idea is to obtain a recovery protocol for linear interaction replication protocols which minimizes the effort and cost of the recovery process, without stopping the replicated system work for *primary partitions*. It is also intended to perform partial recoveries, when needed. Finally, as our design is performed as a middleware recovery system, it can be easily applied to different transactional scenarios, specially including database replicated systems. The obtained results can be used to perform a generic revision of recovery protocols for constant interaction replicated systems.

This paper is structured as follows. Section 2 details the assumed system model, the node system architecture, the assumed failure model, as well as the associated progress condition. The general recovery schema is described in section 3. Afterwards, the information needed by the recovery process is explained in section 4. The following sections, 5, 6 and 7, present three important aspects: the *amnesia recovery* for realistic systems, the consistency problem due to on-going transactions, and the recovery information persistence policy respectively. Finally, some related work is given in section 8, and section 9 concludes the paper.

2 System Model

Our model considers the replicated system as a group of several replicas, where each replica is located in a different node. These nodes belong to a partially synchronous distributed system: their clocks are not synchronized but the message transmission time is bounded. Each replica has a copy of the whole state system.

The replicated system uses a group communication system (*GCS*) providing different communication semantics. Point-to-point and broadcast deliveries are supported. The minimum guarantee provided is a FIFO and reliable communication.

It is also assumed the presence of a group membership service, who *knows* in advance the identity of all potential system nodes. These nodes can join the group and leave it either explicitly or implicitly by crashing. The group membership service combined with the *GCS* provides *Virtual Synchrony* [9] guarantees, thus each time a membership change happens, it supplies consistent information about the current set of reachable members. This information is given in the format of *views*. Sites are notified about a new view installation with *view change events*.

As shown in figure 1, the replication protocol performs its linear interaction by broadcasting (typically with an abcast) the update operations among the *application view* members. Also the membership service collaborates with the communications system to detect changes in the group composition and notify these changes to the group members. Finally, the *recovery protocol* utilizes the membership service to detect outdated nodes and the communications system to send recovery information. Also, the recovery protocol uses the membership service for triggering the process.

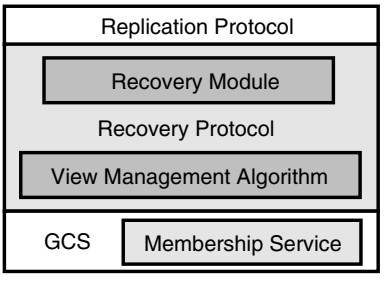


Fig. 1. Node Architecture

The view notification mechanism is extended with node application state information providing the *enriched view synchrony* [10] approach. This makes simpler and easier the support of system cascading reconfigurations. These enriched views (*e-view*) not only inform about active nodes, but they also inform about the state of active nodes: outdated or up-to-date. The use of e-views refines the *primary partition* model into the *primary subview* model, therefore the system only can work when a *progress condition* is

fulfilled[1]. Parallelly, the state consistency is ensured because only the primary *subview* is able to work in partition scenarios. Thus, this subview is the only one allowed to generate recovery information, which will be afterwards used for recovery. For simmilar reasons, new transactions only can be started by fully updated nodes.

2.1 Failure Model

We consider the *crash-recovery with partial-amnesia* model instead of the crash or fail-stop model[2] for node failures. This implies that an outdated node must be recovered from two "different classes of *up-to-dateness* losses": forgotten state and missed state. This assumption supports a more realistic and precise way to perform the recovery process. So the assumed model allows to recover failed nodes from their previous crashing state maintaining their assigned node identifiers. Consequently, when a node crashes, every active node must abort any transaction started by the failed node whose commit messages have not been yet delivered. A similar behavior is adopted when the system can not go on because the progress condition has been lost. In this situation, the nodes in minority (e.g. disconnected) must also abort the started transactions whose commit message has not been yet delivered. Thus, the whole activity that was not committed during the working life is aborted.

2.2 Progress Condition

A *progress condition* determines when the replicated system can work. This condition must be fulfilled in order to avoid the existence of majority partitions that, if they go on working, could lead to different information evolutions in the replicated system. This definition is extended in order to guarantee that a majority partition will be always able to reach a consistent up-to-date state for every composing node. The selected progress condition influences the recovery information needed by the recovery protocol. In order to define these conditions a replicated system compound by n replicas is assumed.

The most traditional condition consists of requiring $\frac{n}{2} + 1$ *up-to-date nodes*. Thus, this condition will not let the system work until $\frac{n}{2} + 1$ nodes are fully up-to-date, even if the partition contains N alive nodes. When this condition is required, in any possible majority partition it will exist at least one node fully updated able to recover out-of-date nodes, regardless the failure history.

A less restrictive condition consists of requiring $\frac{n}{2} + 1$ *alive nodes* to conform a majority view. It lets the replicated system to work as soon as a majority partition is achieved. However, since only up-to-date members are enabled to start transactions, and a majority partition could contain no up-to-date node when this second progress condition is assumed, it becomes possible for a majority partition to be unable to progress if a cooperative recovery [2] is not guaranteed to be feasible. To this end, it must be collected specific recovery information with particular policies.

[1] This characteristic prevents the system from working in the starting phase until a primary subview is reached, and therefore, during this initial phase, the recovery protocol must not perform any work.

[2] This information can be spread among all alive nodes, so they must work together to reach the last system data state.

3 Log-Based Recovery

Since the replication protocol we assume uses linear interaction for implementing its functionality, the most natural way for performing the recovery will follow a log-based strategy to recover outdated nodes.

The main dependency of using log-based recovery relies on the way the upper replication protocol broadcasts the transaction updates among replicas. Any recovery protocol extracts from the broadcast messages the recovery information. However, while constant interaction protocols have typically a single message for each transaction, linear replication protocols manage multiple messages per transaction. Having multiple messages per transaction will present several complications for the recovery protocol, being shown the two more important in sections 6 and 7. Moreover, different storing and recovery policies can be applied, being presented here the simplest one.

The log-based node recovery process is initiated by a node detected to be outdated after its (re)connection to the replicated system. The outdated node starts selecting a *recovery master node* (RMN), which must be one of the most up-to-date alive nodes, and performs the following steps:

- *missed recovery stage (MRS)*. Where outdated nodes update their missed views.
- *current view recovery stage (CVRS)*. This last step is done if the replicated system is working during the outdated node recovery. It is performed once the node has applied all its missed views, and its goal is to apply in the recovering node any message received during the previous stages (which could not be applied, since the node was not updated yet).

In addition, the (re)connected node must always do the *amnesia recovery stage* (ARS) priorly to any other stage. Notice that the ARS is always triggered by node (re)connections. The used log-based recovery algorithm is summarized in figure 2.

```
if the outdated node is the (re)connected node:
    it performs its ARS
the outdated node performs its MRS:
    for each non-applied view v_i:
        if the outdated node does not have the log-recovery view information for v_i:
            it demands the log-recovery information for v_i to the RMN(v_i)
        the outdated node updates view v_i
        notify the alive nodes that v_i has been recovered in this node
if the replicated system was working during the node recovery (all lost views were applied):
    the outdated node performs its CVRS
```

Fig. 2. Recovery Algorithm.

The first step, the ARS, is used by (re)connected nodes to retrieve the right state they had at their crash time. In this process, the outdated node must apply the messages received and not yet committed before its failure. A deeper discussion of this recovery step is done in section 5.

The *MRS* stage is performed by every outdated node. Each outdated node starts the recovery of its missed views by selecting its RMN for such views (in a primary partition, a single node can be used for every view[3]). Since views are sequentially numbered, the first view to be recovered is the next one to its last fully applied view, and finishes when it reaches the last applied view in the RMN[4]. As this recovery is performed view by view, it is possible to request to the RMN just the log-recovery information for the lost views. Once the outdated node has this information (i.e. missed messages), it applies them to recover this view. The application of these missed messages must be performed in the same order they were delivered in the replication system total order. When the outdated node has finished the recovery of this view it must notify all alive nodes that it has recovered this view, therefore they may discard the available recovery information for such view.

Every time a membership change occurs it must be checked how it affects to each recovery process started. Obviously if the outdated node crashes the recovery process ends. If the RMN has failed the outdated node must look for a new RMN for going on with the recovery process.

The third step, *CVRS*, is performed by outdated nodes recovered as long as the system is working (i.e. if the (re)connection was into a primary partition). In this scenario, the system generated activity during the recovery process, but the recovering node delayed the application of such activity (it just persisted it in a queue as part of a "seen view"). Thus, once all the non-previously-applied views are applied in the recovering node, it must conclude its recovery by applying all these delayed messages in their delivery order. If a new working view is installed during the recovery of a node, a new queue is created for the new view[5].

The generic log-based recovery protocol, as it has been described, is intended to provide recovery support for replication systems based on linear interaction. But the use of linear interaction rises several problems that must be considered for recovery purposes. As we show in section 6, the consistency problem associated to ongoing transactions must be emphasized as one of the main problems to be treated.

4 Recovery Information

As said, we assume a *crash-recovery with partial-amnesia* failure model. Consequently, once a crashed node is reconnected, it neither knows anything about the work performed during its disconnection nor can remember exactly which was its exact state before the crash occurrence.

Thus, node recovery must include both the recovery of missed state, and the recovery of the "forgotten state". The first one refers to the data state that the node has not received because it was disconnected. The second one covers the state received but later lost (due to the amnesia) when the node crashed. Therefore, the recovery system must maintain information allowing it to handle these two out-of-date causes.

[3] This approach allows *partial recoveries* if the outdated node is in a system which does not fulfil the progress condition.

[4] If the system is in a working view the previous view to the current one is considered the last one applied in the RMN.

[5] Notice that in this case the outdated node has already the information of the previous view.

The *log-based* policies use the broadcast messages as recovery information. If the amnesia is not considered, the recovery information only refers to messages missed by non-connected nodes. Thus, in this case, the recovery information must be created when the membership monitor detects non-connected nodes in the system. The information must be maintained until the outdated nodes have applied the missed messages.

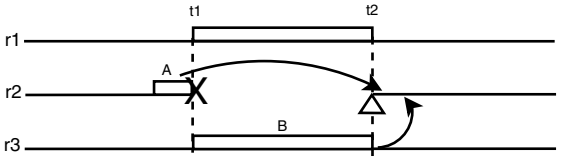


Fig. 3. Log Recovery Information.

However, the amnesia recovery information for *log-based* policies relates to those messages belonging to transactions not-yet-committed at crashed nodes when they failed. Therefore, the recovery system must store the received and not-yet-committed messages at each node. These circumstances require from each node to maintain its own log-based amnesia recovery information during its normal activity, whilst the rest of recovery information is maintained in other nodes just when failed nodes exist.

Figure 3 shows the information needed to recover an outdated node using the log-recovery strategy. In this figure node $r2$ crashes at time $t1$ and reconnects at time $t2$. At this moment, the system must start the $r2$ recovery. Firstly, it needs the A block information recovery used to recover the amnesia, it contains the messages received but not committed by $r2$ before its crash at $t1$. Secondly, the system needs B block which contains the messages missed by $r2$ (either committed or not-yet-committed) during its failure time. Obviously, the A block can be managed and maintained by $r2$ whilst the B block must be generated and managed by a non-failed node.

Complimentary to the above information the system must maintain information about which nodes have missed which views. Thus, each time a new view is installed in the system the alive nodes must store the identifiers of the failed nodes in this view. When a node recovers a view its identifier is deleted from the recovered view failed nodes information. When the list of failed nodes in a view is emptied, the recovery information of this view can be deleted.

5 Amnesia Recovery

As described above, the assumed failure model implies that on reconnection of crashed nodes, they can not remember exactly which was their last state before the crash occurrence by themselves. Thus, extra information is needed to be maintained to this particular end, being afterwards used in ARS.

The amnesia problem is manifested in different ways. The main amnesia effect refers to the not-yet-committed transactions state which is lost when a node crashes. This state must be recovered before performing the real recovery process in order to reach consistent and non-diverging data states. This lost state can be recreated by reapplying the messages belonging to the on-going transactions existing at the failure time. Therefore, to perform this amnesia recovery, the system must maintain two different types of information. On one hand, the system must store the messages belonging to non-committed transactions in order to reapply them when the node is reconnected. On the other hand, the node must know precisely the identity of every committed transaction in the underlying system of this replica. This is necessary because it is possible for a transaction expected to be committed in the system to have not been actually committed in this replica (e.g. due to overhead or because the node is being recovered and could not apply the currently received replication messages). Thus, this amnesia phenomenon can be observed at two levels:

The amnesia at the *transport level* implies that received messages non-persistently stored are lost when the node crashes. If this occurs, the amnesia recovery could not be performed. So the system must ensure these messages are available for recovery purposes by storing them in a persistent way.

As long as there exist failed nodes in the system, the alive nodes must persistently store the broadcast messages in order to perform the recovery of these failed nodes, as it has been commented in 7. Thus, this information can be used in ARS. However, if there are no failed nodes, each node can manage its own amnesia recovery information persistently. This storage of received messages must be kept until the respective owner transaction is either committed or aborted[6]. Moreover, if the messages must be maintained after its transaction commit, they must be marked in some way to remember that they have been already applied.

In addition, the permanent storing of a received message must be performed atomically with the GCS message delivery. Under this principle, if the node is not able to persistently store the message, the GCS does not consider this message as delivered (thus ensuring that the delivery is coupled with the persistent storage).

If other approaches are taken (i.e. maintaining the information in other nodes) messages cannot be discarded without additional synchronization rounds, because the *other* nodes do not know by themselves when a message could be discarded. This implies a high memory cost, or network overheads.

The amnesia at the *replication level* relates to the fact that the system can not remember which were the really committed transactions. Even for those transactions for which the "commit" message was applied, it is possible for the system to fail *during* the commit. Thus, the information about the success of such commit must be also stored because it is needed by the recovery amnesia process in order to know which are the messages that must be applied (as we discussed previously). So, at the replicated system level, the problem is to know if a "commit" message was successfully applied before the failure or not.

[6] i.e. discarding them on transaction aborts, or maintaining them for committed transactions *only* when failed nodes exist.

The mechanism for generating this information consists of maintaining some information about the last committed transaction for each open connection. Thus, when a transaction commit is performed in the replica, the system must write this information in a single atomic step, as part of the transaction itself. Thus, on commit success, the system contains the identity of this last committed transaction. Afterwards, when the connection is closed, the entry corresponding to this connection is erased.

This information is useful in the recovery process to check if messages marked as not committed have been really committed. When the node becomes alive again and starts its ARS it will check if there are messages marked as not committed, but its owner transaction is marked as committed in the replica.

A similar problem arises regarding the state associated to not committed messages (messages belonging to not yet committed transactions), since it is lost at the crash instant, since the replication system is also a transactional system. Therefore, these messages applied by the replication system but not committed must be again reapplied.

There are three possible scenarios where messages maintained as non-committed belong to a transaction whose owner connection does not have an entry in the table of committed transactions:

- *The node did not start to apply this connection and its transactions before the node crash.* Then, these messages must be applied in the ARS.
- *The connection is closed before committing some of its related transactions (implying that these transactions will be aborted) but the node crashes before the system erases those messages.* Thus, all the messages will be reapplied in the ARS but they will be aborted again as it has happened in the normal work way.
- *The transaction was really committed, the system has not marked yet its messages as committed, and it has already deleted its table connection entry.* This scenario must be avoided, because it will lead the system to apply twice a transaction if a recovery process is performed. So, when the "remove connection" message is applied, the removal of the corresponding entry in the system must be done after the middleware considers committed the transaction.

As a result, the *ARS* recovery stage will just consist of reapplying the messages marked in the log recovery as "not applied" or "not committed", first checking against the replica if they were not really committed.

6 On-Going Transactions and Consistency

The use of linear interaction in replication protocols implies the broadcast of messages belonging to not-yet-committed transactions. Thus, these messages belonging to different transactions are interleaved and applied to the replica in their delivery total order. Finally, each transaction is committed when its commit is applied. In this context, if a node crashes, all associated changes to not-yet-committed transactions are lost whilst associated updates to committed transactions remain permanent.

Afterwards, when the crashed node becomes again active, the recovery process updates it, reapplying among others the messages associated to not-yet-committed transactions at the crash time, while the committed transaction messages at the crash time are

non reapplied (since they were already persisted in the replica). In this scenario, some inconsistencies could arise if these reapplied messages were interleaved with committed transaction messages in the original work sequence, because this original order is misunderstood in the recovered node. The inconsistencies appear if these transactions conflict and the selected isolation level tolerates these conflicts.

It must be noticed that this problem only occurs when an outdated node reconnects to the replicated system, and this last one has been working continuously during the disconnection of the recovering node.

The following example shows this problem in a more intuitive way. Let us assume a replicated system of three nodes, $\alpha = r_1, r_2, r_3$. At the beginning, the three nodes are up-to-date and working. During a replicated system work lifetime period the sequence of events shown in the figure 4 happen.

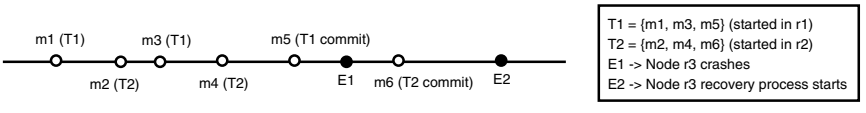


Fig. 4. Timeline events

As it could be seen, in the original sequence order messages of $T1$ and $T2$ are interleaved. The $T1$ commit is performed before the crash of node r_3 while the $T2$ commit is done during the r_3 failure time. Therefore the final messages sequence seen in r_1 and r_2 is $[m1, m2, m3, m4, m5, m6]$, whilst the final message applied sequence in r_3 once it has been recovered is $[m1, m3, m5, m2, m4, m6]$.

This message order misunderstood in r_3 is created by the recovery protocol. In fact, the node r_3 before $E1$ applies the same sequence message order as r_1 and r_2, $[m1, m2, m3, m4, m5]$. However, when it fails, it loses non-committed changes (in this case the changes performed by $T2$). When r_3 reconnects to the system its data state is $[m1, m3, m5]$

At this moment the recovery process applies the not-yet-applied updates in r_3, which are the messages regarding $T2$, which provokes the message order misunderstood. This different message order in $T2$ could lead to a different data state with regard to the state in r_1 and r_2 if $T1$ and $T2$ conflict and the selected isolation level tolerates it (for instance, when using *Snapshot Isolation*). In this example a conflict could arise if $m2$ and $m3$ perform the following sentences respectively:

m2→ "UPDATE employees SET salary=salary*1.05 WHERE points>10"
m3→ "UPDATE employees SET points=points+1 WHERE points == 10"

With these sentences it is possible that in r_3 some employees increase their salary while in r_1 and r_2 their salaries are not increased. Thus the recovery protocol can generate different data state evolutions in recovered nodes with regard to not recovered nodes. This problem appears because the recovery protocol cannot store the original context of *on-going-transactions*.

In order to avoid this problem two solutions can be applied. The first and more natural one would be to select an isolation level that aborts this kind of conflicts, which in fact implies to apply the SERIALIZABLE isolation level. Thus, this approach avoids the problem presented above allowing to use the proposed recovery strategie.

Other option would be to relax the required consistency guarantees, which tolerates this kind of conflicts, but avoiding the above presented problem performing the recovery process under a *special condition*. This condition requires that the recovery process must be done when the recovery messages to apply (i.e. on-going transactions) does not conflict with transactions committed during its life. It means that the recovery messages to apply were not interleaved with conflicting committed messages. As controlling the fulfilment of this condition is difficult it must be selected an easiest control condition. This new condition would be to select as base recovery point (*BRP*) a "timepoint" in the replicated system lifetime where there does not exist on-going transactions. Obviously this BRP must be later than the moment when the outdated node crashed. Thus, the outdated node recovery is performed in two steps: In the first one the outdated node recovers the data state up to the selected BRP [7]. In the second step the messages delivered after the selected BRP (if there exist) will be applied using the log-based approach. This solution could be implemented in two different ways: reactive (forcing the existence of a BRP after the reconnection of the recovering node) and proactive (founding a suitable BRP during the normal work of every node).

As conclusion, the only possible solution to use a log-based recovery approach without using a version-based one forces the system to adopt the SERIALIZABLE isolation level. Another aspect that must be considered is which recovery information policy must be applied, and how it is affected by the linear recovery interaction. The following section is devoted to the discussion of this aspect.

7 Recovery Information Persistence

There exist some recovery situations, that depending on the progress condition and the persistence policy for the recovery log-information adopted the system is unable to guarantee the correct system data state progress.

Obviously, the messages of committed transactions must be persistently stored to perform the recovery of failed nodes.

The further question is if messages belonging to ongoing transactions must be also persisted in all nodes. The answer depends on the adopted progress condition. For instance, assume that the used progress condition is the *majority partition* and that messages not committed are not persistently stored. Consider then the following case, a replicated system $\alpha = \{r_1, r_2, r_3, r_4, r_5\}$ which is working with r_1, r_2, r_3 as alive replicas, and there exists a long term transaction T_1 started in r_1 which has already broadcast some operations m_1, m_2. Then, a failed node (r_5) reconnects and the system starts its recovery while T_1 broadcasts more messages (m_3, m_4) being received by all alive nodes. Now, before T_1 commit message is broadcast and r_5 is being recovered, one of the previous alive nodes (e.g. r_3) fails momentarily, excluding the one that

[7] It must be remarked that in this step it can not be used the log-based recovery strategy because the problem of different state evolution would not be avoided.

started T_1 (if r_1 crashes T_1 is aborted), and reconnects quickly. Thus r_4 will lose the messages belonging to T_1. At this moment, after the r_3 crash and reconnection, and before the r_3, r_5 full recovery, the T_1 commit is broadcast. Then, T_1 is committed in r_1, r_2 while r_3, r_5 maintain this message to apply it afterwards because they have not yet been fully recovered. Immediately, r_1 and r_2 crash before the recovery system has transferred m_1, m_2 to r_3, r_5. Thus, it would not be possible to reconstruct T_1 in r_3 and r_5. Moreover, if the next majority partition is reached because $r4$ reconnects, the system can not progress in an accurate manner because it will not be able to commit T_1 in r_3, r_4, r_5 since it has been committed in r_1, r_2. Notice that this second case is also related to the amnesia recovery process of node r_3.

This case of non correct progression can be avoided if the selected progress condition only lets the system to go on working if $\frac{n}{2} + 1$ replicas are fully recovered instead of going on working each time a majority partition has been reached. In this case this proposed progress condition will abort T_1 when the node r_3 has crashed.

The other solution that can be adopted is to maintain the *majority partition* as progress condition, but forcing the recovery system to persist messages belonging to ongoing transactions. With this solution r_3 would not have lost the m_1, m_2 messages when it crashed, being afterwards the recovery system enabled to reconstruct T_1 without the participation of the nodes that have committed it.

Between these solutions, the second one is selected because it relaxes the progress condition enabling the system to work any time a *majority partition* is reached. Evidently, this adopted recovery system presents the necessity of storing permanently broadcast messages as a drawback.

8 Related Work

In the area of recovery protocols for replicated distributed systems two basic approaches are used: version based and log based. The first one consists of transferring to outdated nodes those data items changed during their failure period, whilst the second one consists of transferring the messages missed by outdated nodes.

A wide range of proposals about this classic problem[11] have been presented for a long time in the last years either version-based [3], [12] and log-based [3], [8], [7]. First ones are typically useful for long-term outages whilst the latter ones present better performance for recovering short-term failures. Therefore, combining a version-based technique with a log-based one to construct a recovery framework has been proposed in several works as [3], [8] to improve the recovery features, choosing the recovery strategy that presents a lower cost each time an outdated node is detected.

The most widely assumed correctness criterion for replicated systems is *1-copy-serializability*, which consequently leads to recovery protocols intended to work with such systems, often using log-based approaches [7], [3], [4]. However, the use of other isolation levels has not been traditionally treated in recovery protocols, probably based on the assumption that replication protocols are intended to provide *1-copy-serializability*. In fact, this is the isolation level that best fits the consistency guarantee in a general distributed system. But, when the replicated state requires high transfer rates, its use implies a high performance cost. Also, for transactional systems, where

isolation must be enforcing by using specific concurrency control mechanisms, this problem is even worst. These two drawbacks are specially problematic in replicated databases, where the enforcement of *1-copy-serializability* usually leads to extremely inefficient systems. Therefore, relaxed isolation guarantees are used there to alleviate the performance degradation associated to the highest isolation level. One of the most widely adopted relaxed levels is *Snapshot Isolation*, having the interesting property of allowing read-only transactions to proceed without being blocked or delayed by any other transaction. In this way, recent publications [13], have proposed some replication protocols providing *Snapshot Isolation* [14]. Moreover, the most extended DBMS (Postgres, MySQL,..) provide snapshot isolation as the basic isolation guarantee.

On the other hand, recovery protocols are also typically designed to work for replicated protocols based on *constant interaction*[3]. Others ,simply outline how these protocols can work using linear interaction. In fact, a few works have designed recovery protocols[4] which work over linear-interaction-based systems. In [4], different log-based recovery protocols are presented including proposals either for constant and linear interaction, but always focused on SERIALIZABLE systems.

9 Conclusions

In this paper, we detail a middleware-based general log-based recovery strategy intended to provide fault tolerance support for *linear interaction*-based replication systems. This obtained system lets to perform on-line recoveries, fulfilling one important condition for building a high available system. Most important, this paper studies which effects has the use of *linear interaction* on the recovery work, specially emphasizing the global data state consistency and the recovery information management.

Moreover, the paper also analyses and designs an amnesia recovery process as part of the whole recovery strategy, supporting a more realistic failure scenario. This amnesia phenomenon has been discussed at the two different levels in which it could appear, and a basic strategy to bound the amnesia problem has been also detailed.

In addition, the proposed strategy supports re-inclusions in minority partitions, performing partial or full recoveries, helping the system to accelerate outdated nodes recovery.

Other important point considered in the paper is the progress condition, meaning the requirements that the replicated system must fulfil for continuing working. Two different progress conditions have been discussed: One following the stricter [3] approach and another more relaxed, based on the majority partition concept. This second progress condition, as it has been shown in section 7, must be accompanied by a particular recovery information policy, required to avoid possible different state evolutions, depending on the failure histories.

Another important aspect demonstrated in this work, in section 6, is that using a linear-interaction replication protocol forces the system to use SERIALIZABLE isolation level to avoid consistency problems after any log-based recovery process. In fact, the use of any other isolation level could let different replicas to reach different data states after applying the same transactions set.

In an indirect way, this paper also has highlighted that the existence and management of on-going transactions (due to linear interaction) from a recovery point of view

presents several difficulties (replicated consistency, amnesia delimitation boundaries), whose solution reduces the whole system performance and scalability. Therefore this paper reinforces the traditional arguments (traffic net overhead) that discourage the use of the linear interaction approach on replication systems.

A sequel of this work will be a generic revision of existing recovery protocols based on constant interaction taking under account the results obtained in this work for recovery systems working in linear interaction replicated systems.

References

1. Wiesmann, M., Schiper, A., Pedone, F., Kemme, B., Alonso, G.: Database replication techniques: A three parameter classification. In: SRDS. (2000) 206–215
2. Hadzilacos, V., Toueg, S.: Fault-tolerant broadcasts and related problems. In Mullender, S., ed.: Distributed Systems. 2nd edn. ACM Press (1993) 97–145
3. Kemme, B., Bartoli, A., Babaoğlu, O.: Online reconfiguration in replicated databases based on group communication. In: Intl.Conf.on Dependable Systems and Networks, Washington, DC, USA, IEEE Computer Society (2001) 117–130
4. Holliday, J.: Replicated database recovery using multicast communication. In: NCA, IEEE Computer Society (2001) 104–107
5. Wiesmann, M., Schiper, A.: Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng. **17**(4) (2005) 551–566
6. Ricciardi, A.M., Schiper, A., Birman, K.P.: Understanding partitions and the "no partition"" assumption. Technical report, Ithaca, NY, USA (1993)
7. Jiménez-Peris, R., Patiño-Martínez, M., Alonso, G.: Non-intrusive, parallel recovery of replicated data. In: SRDS, IEEE Computer Society (2002) 150–159
8. Castro, F., Esparza, J., Ruiz, M., Irún, L., Decker, H., Muñoz, F.: CLOB: Communication support for efficient replicated database recovery. In: 13th Euromicro PDP, Lugano, Sw, IEEE Computer Society (2005) 314–321
9. Birman, K.P., Renesse, R.V.: Reliable Distributed Computing with the ISIS Toolkit. IEEE Computer Society Press, Los Alamitos, CA, USA (1993)
10. Babaoğlu, O., Bartoli, A., Dini, G.: Enriched view synchrony: A programming paradigm for partitionable asynchronous distributed systems. IEEE Trans. Comput. **46**(6) (1997) 642–658
11. Bernstein, P.A., Hadzilacos, V., Goodman, N.: Concurrency Control and Recovery in Database Systems. Addison Wesley, Reading, MA, EE.UU. (1987)
12. Castro, F., Irún, L., García, F., Muñoz, F.: FOBr: A version-based recovery protocol for replicated databases. In: 13th Euromicro PDP, Lugano, Sw, IEEE Computer Society (2005) 306–313
13. Lin, Y., Kemme, B., Patiño-Martínez, M., Jiménez-Peris, R.: Middleware based data replication providing snapshot isolation. In Ozcan, F., ed.: SIGMOD Conf., ACM (2005) 419–430
14. Berenson, H., Bernstein, P.A., Gray, J., Melton, J., O'Neil, E.J., O'Neil, P.E.: A critique of ANSI SQL isolation levels. In: SIGMOD Conf., ACM Press (1995) 1–10

Mobile Agents Based Collective Communication: An Application to a Parallel Plasma Simulation

Salvatore Venticinque[1], Beniamino Di Martino[1], Rocco Aversa[1], Gregorio Vlad[2], and Sergio Briguglio[2]

[1] Dipartimento di Ingegneria dell' Informazione - Second University of Naples - Italy
Real Casa dell'Annunziata - via Roma, 29 81031 Aversa (CE) - Italy
[2] Associazione EURATOM-ENEA sulla Fusione, C.R. Frascati, C.P. 65, 00044, Frascati, Rome, Italy

Abstract. Collective communication libraries are widely developed and used in scientific community to support parallel and Grid programming. On the other side they often lack in Mobile Agents systems even if message passing is always supported to grant communication ability to the agents. Collective communication primitives can help to develop agents based parallel application. They can also benefit social ability and interactions of collaborative agents. Here we present a collective communication service implemented in the Jade agent platform. Furthermore we propose its exploitation to interface transparently heterogeneous executions instances of a scientific parallel application that runs in a distributed environment.

1 Introduction

Collective communication is a communication activity that involves more entities belonging to a group and sharing a common goal or a common interest. In commercial application it can be exploited to enhance the way internet users have today to communicate by Internet. In high performance computing it represents a parallel programming paradigm that allows communication and synchronization of processes by message passing. Collective communication primitives are constructs of the language which exploit the underlaying middleware to perform specified schemes of message exchange. They can help to develop agents based parallel applications. They can also benefit social ability and interactions of collaborative agents. Mobile Agents technology has been widely addressed in the scientific community to provide a flexible programming approach in parallel and distributed environments. Some experimental results can be found in [1,2,3,4]. A mobile agent is a Software Agent with an added feature: the capability to migrate across the network together with its own code and execution state. This paradigm allows both a pull and a push execution model [5]. In fact, the user can choose to download an agent or to move it to another host. Mobility can provide many advantages when we aim at developing distributed applications. System reconfiguration by agent migration can help to optimize

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 724–733, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the execution time by reducing network traffic and interactions with remote systems. Furthermore, statefull migration allows to redistribute dynamically the agents for load balancing purposes. Several different criteria can guide agent distribution, such as moving the execution near to the data, exploiting new idle nodes, or allocating agents on the nodes in such a way that communications are optimized. Many different implementations of the mobile agent programming paradigm are available. A list of more than 60 known agent platforms can be found at `http://labe.felk.cvut.cz/~bendap1/agentsoftware/list.html`.

However existing Mobile Agents platforms do not provide collective communication primitives which can help the programmer to develop parallel applications. Here we present a collective communication service implemented in the Jade agent platform and we propose its exploitation in a scientific parallel application to interface transparently heterogeneous executions which run in parallel in a distributed environment. Next section introduces message passing, collective communication concepts and technology together with agents systems. Section three presents the collective communication service and its implementation. Section four describes the parallel application we are going to use as case study. Finally Conclusions summarize the current status of our research and ongoing work.

2 Message Passing and Collective Communication: Concepts and Technology Together with Agents Systems

Message passing libraries and collective communication primitives to support parallel and Grid programming are widely developed and used in scientific community. As an example, we cite MPI (Message Passing Interface), the best known standard [6] which defines more the 300 routines. Its implementations for fortran and C languages are available on different architecture by countless developers. JavaMPI [7] is a pure java implementation of the standard, while MPIJ [8] is a java interface to a native code implementation of the supported methods. Other parallel environments such as DECK [9] provide similar facilities. On the other side collective communication primitives can be rarely found implemented in agents platforms. The FIPA standard [10] provides specifications to design an agent platform. It describes the software architecture of an agent platform, the basic services which must be provided, the life cycle of an agent, but the most relevant part of the standard concerns the agent communication language. In fact, FIPA original effort aims, above all, at supporting interoperability among autonomous agents belonging to the same platform or to different ones. Regarding agent communication FIPA defines communicative acts and their requirements [FIPA00037], grammatical structure of Agent Communication Language, agent interaction protocols, their ontology, semantic and requirements. The standard do not deal with the transport communication protocol and the way to exchange messages. Nowadays many platforms are compliant with the FIPA specifications and implement many message transport protocols (MTPs) to support

interoperability with other agent systems. An example is JADE, developed by TILAB (Telecom Italia laboratory) which implements HTTP MTP, IIOP MTP, Orbacus MTP for inter platform communication and RMI for intra platform communication. Regarding the communication primitives supported by a mobile agents platform different message passing models can be available: synchronous, asynchronous, future messaging, etc. They differ according to the way the sender behaves when sending a message. It can wait for the end of the message handling at destination, it can wait or check later for a response. The Aglet Workbench, developed by IBM Japan research group [11] supports all the mechanisms described above.

3 Mobile Agents Collective Communication

In a previous paper we have presented MAGDA - Mobile Agents based Grid Architecture - [14]. It is conceived in order to provide secure access to a wide class of services in an heterogeneous system, locally and geographically distributed. MAGDA is a layered architecture, which strictly adheres to the Layered Grid Model[15]. We exploited MAGDA to develop Mobile Agents based parallel applications in [12,4,13]. We extended the multicast communication mechanism supported locally in the Aglet Workbench in order to make it working in a distributed environment. Now we aim at providing at agent level a collective communication library and at developing an agent gateway to extend the communication ability of legacy applications (such as classical MPI applications) to heterogeneous and distributed environments. The conceived model defines two communication layers, as shown in Figure 1. The first one is implemented by a native parallel middleware; the second one, at upper level, is implemented by agents. An agent gateway is able to perform as a bridge forwarding messages from an environment to the other. A similar approach is exploited by MPICH_G2 [16] in the Globus middleware among proprietary implementation of the MPI standard. In our architecture, a collective communication manager coordinates groups, collects and delivers messages being able to migrate across the network, in order to optimize the mean latency and the traffic according to the physical location of the communicating parties.

3.1 Designing the Collective Communication Service

We designed the collective communication facility as a service provided by an agent inside the Jade platform. The provider is both a group and a message manager. It can be federated with other providers in order to distribute the handling of groups and of messages. Furthermore it is mobile and can migrate to a best suited node according to the physical distribution of group members in order to minimize latencies and traffic which affect the network. The communication manager accepts subscriptions by agents to new or to already available groups. Afterwards, agents can send to it messages which must be handled according to the selected communication mechanism. In the starting phase the provider

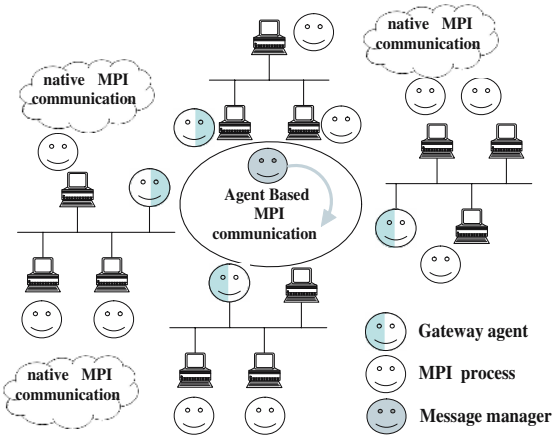


Fig. 1. The two layer communication hierarchy

publishes its service as Collective Communication Service in the directory facilitator which is the FIPA standard directory where agents look for available services (according to the standard, it must be implemented by an agent and must run in each platform). Each agent requestor is able to look for the service and to subscribe dynamically to a collective communication group identified by the provided identification label. If the required group is unknown a new group is built with the received label otherwise the agent is added to the already existing group. The agent subscription is delayed till any open collective communication involving that group will be successfully completed. Each agent requestor is able to subscribe to a group, to unsubscribe to the same group and to take part in a collective communication by sending a message to the provider and waiting for the response. According to the selected communications mechanism, the response is returned when all the other group members have joined the same communication instance. We support the following communication mechanisms:

- *broadcast*: the message is forwarded to all the members of all existing group registered by a provider
- *multicast*: the message is forwarded to all the agents of the specified group.
- *scatter*: the content of the message received by a specified agent (the root) is distributed to the other group members.
- *gather*: the content of the received messages are collected in a vector that is sent to the specified root member of the specified group. All the others agent receive an acknowledge when the communication is completed.
- *allgather*: the content of the received messages are collected in a vector that is sent to all the members of the specified group.
- *alltoall*: content of the received message from each agent is divided among the others group members in the way that each one receives part of the content from all messages.

We implemented the described service in the Jade platform. We provide a set of APIs at client side that allows the programmer to create a MPI message and to invoke the desired collective communication primitive. In the jade programming model the developer can define an agent behavior extending a basic *Behavior* class whose instances will be scheduled by the agent itself. As it is shown in Figure 2 we implemented a MPIBehaviour that looks for an available collective communication service and provide some final methods which allow the agent to subscribe/unsubscribe to communication groups and to take part in the supported collective communication actions. The datatype parameter identifies

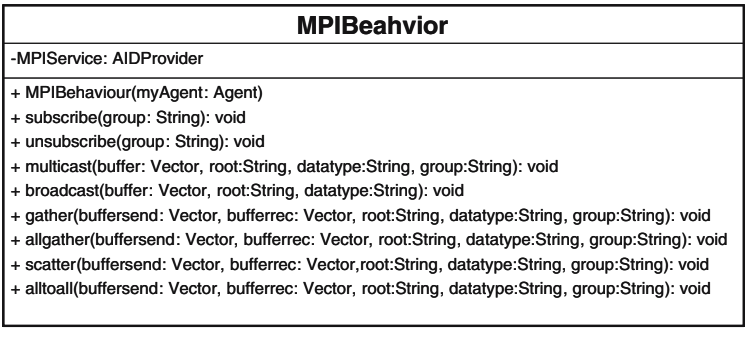


Fig. 2. The MPIBehaviour class diagram

the object class inserted in the buffer. The programmer can send or receive any datatype and any amount of object by a Vector instance. The root parameter is the agent identifier of the receiver/sender in the specific communication pattern. The group parameter defines who will participate to the communication.

3.2 Mobile Agent Communication Gateway

In order to interface the agent based communication service described in the previous section with legacy MPI applications we have designed an agent gateway. The agent will start the MPI application and will be considered by the MPI processes as a process itself. When a communication primitive is invoked, the agent transparently will act as message gateway forwarding the information to other agent gateways which subscribed to a collective communication group of higher level. Hence, the agent gateway represents a virtual process which is physically allocated on a remote heterogenous node or distributed among more nodes. It will be implemented by a real MPI process that will take part in the communication at two different levels: at the lower one it exploits native MPI, at the higher one the agent technology. We are still evaluating its implementation details so we are not able to provide a final design of the component of the communication middleware. In [17] we have implemented in a Grid environment a parallel application to study plasma turbulence. Starting from this previous

experience, we are going to apply the presented approach to the case study described in the next section. As shown previously, we have provided all the necessary communication primitives: scatter, broadcast and alltoall implementations. Note that all the MPI primitives are handled by MPI daemons running on each node. We will have to extends these daemons to support the hierarchical communication presented above.

4 Plasma Turbulence Simulation

Particle-in-cell simulation consists [18] in evolving the coordinates of a set of N_{part} particles in certain fluctuating fields computed (in terms of particle contributions) only at the points of a discrete spatial grid and then interpolated at each particle (continuous) position. Two main strategies have been developed for the workload decomposition related to porting PIC codes on parallel systems: the *particle decomposition* strategy [19] and the *domain decomposition* one [20,21]. Domain decomposition consists in assigning different portions of the physical domain and the corresponding portions of the spatial grid to different processes, along with the particles that reside on them. Particle decomposition, instead, statically distributes the particle population among the processes, while assigning the whole domain (and the spatial grid) to each process. As a general fact, the particle decomposition is very efficient and yields a perfect load balancing, at the expenses of memory overheads. Conversely, the domain decomposition does not require a memory waste, while presenting particle migration between different portions of the domain, which causes communication overheads and the need for dynamic load balancing [22,21]. The typical structure of a PIC code for plasma particle simulation can be represented as follows. At each time step, the code

1. computes the electromagnetic fields only at the N_{cell} points of a discrete spatial grid (*field solver* phase);
2. interpolates the fields at the (continuous) particle positions in order to evolve particle phase-space coordinates (*particle pushing* phase);
3. collects particle contribution to the pressure field at the spatial-grid points to close the field equations (*pressure computation* phase).

We can schematically represent the structure of this time iteration by the following code excerpt:

```
call field_solver(pressure,field)
call pushing(field,x_part)
call compute_pressure(x_part,pressure)
```

Here, `pressure(1:n_cell)`, `field(1:n_cell)` and `x_part(1:n_part)` (with `n_cell`= N_{cell} and `n_part`= N_{part}) represent pressure, electromagnetic-field and particle-position arrays, respectively. In order to simplify the notation, we will refer, in the pseudo-code excerpts, to a one-dimensional case, while the real code refers to a three-dimensional (3-D) application. In implementing a parallel version

of the code, according to the distributed-memory domain-decomposition strategy, different portions of the physical domain and of the corresponding spatial grid are assigned to the n_{node} different nodes, along with the particles that reside on them. This approach yields benefits and problems that are complementary to those yielded by the particle-decomposition one [19]: on the one hand, the memory resources required to each node are approximately reduced by the number of nodes (`n_part`$\sim N_{part}/n_{node}$, `n_cell`$\sim N_{cell}/n_{node}$); an almost linear scaling of the attainable physical-space resolution (i.e., the maximum size of the spatial grid) with the number of nodes is then obtained. On the other hand, inter-node communication is required to update the fields at the boundary between two different portions of the domain, as well as to transfer those particles that migrate from one domain portion to another. Such a particle migration possibly determines a severe load unbalancing of the different processes, then requiring a dynamic balancing, at the expenses of further computations and communications. Let us report here the schematic representation of the time iteration performed by each process, before giving some detail on the implementation of such procedures:

```
 call field_solver(pressure,field)
 call check_loads(i_check,n_part,n_part_left_v,
&    n_part_right_v)
 if(i_check.eq.1)then
  call load_balancing(n_part_left_v,n_part_right_v,
&     n_cell_left,n_cell_right,n_part_left,n_part_right)
  n_cell_new=n_cell+n_cell_left+n_cell_right
  if(n_cell_new.gt.n_cell)then
   allocate(field_aux(n_cell))
   field_aux=field
   deallocate(field)
   allocate(field(n_cell_new))
   field(1:n_cell)=field_aux(1:n_cell)
   deallocate(field_aux)
  endif
  n_cell=max(n_cell,n_cell_new)
  n_cell_old=n_cell
  call send_receive_cells(field,x_part,
&     n_cell_left,n_cell_right,n_part_left,n_part_right)
  if(n_cell_new.lt.n_cell_old)then
   allocate(field_aux(n_cell_old))
   field_aux=field
   deallocate(field)
   allocate(field(n_cell_new))
   field(1:n_cell_new)=field_aux(1:n_cell_new)
   deallocate(field_aux)
  endif
  n_cell=n_cell_new
  n_part=n_part+n_part_left+n_part_right
```

```
endif
call pushing(field,x_part)
call transfer_particles(x_part,n_part)
allocate(pressure(n_cell))
call compute_pressure(x_part,pressure)
call correct_pressure(pressure)
```

In order to avoid continuous reallocation of particle arrays (here represented by `x_part`) because of the particle migration from one subdomain to another, we overdimension (e.g., +20%) such arrays with respect to the initial optimal-balance size, N_{part}/n_{node}. Fluctuations of `n_part` around this optimal size are allowed within a certain band of oscillation (e.g., ±10%). This band is defined in such a way to prevent, under normal conditions, index overflows and, at the same time, to avoid excessive load unbalancing. One of the processes (the MPI rank-0 process) collects, in subroutine `check_loads`, the values related to the occupation level of the other processes and checks whether the band boundaries are exceeded on any process. If this is the case, the "virtual" number of particles (`n_part_left_v`, `n_part_right_v`) each process should send to the neighbor processes to recover the optimal-balance level is calculated (negative values means that the process has to receive particles), and `i_check` is set equal to 1. Then, such informations are scattered to the other processes. These communications are easily performed with MPI by means of the collective communication primitives `MPI_Gather`, `MPI_Scatter` and `MPI_Bcast`. Load balancing is then performed as follows. Particles are labelled (subroutine `load_balancing`) by each process according to their belonging to the units (e.g., the `n_cell` spatial-grid cells) of a finer subdivision of the corresponding subdomain. The portion of the subdomain (that is, the number of elementary units) the process has to release, along with the hosted particles, to neighbor subdomains in order to best approximate those virtual numbers (if positive) is then identified. Communication between neighbor processes allows each process to get the information related to the portion of subdomain it has to receive (in case of negative "virtual" numbers). Net transfer information is finally put into the variables `n_cell_left`, `n_cell_right`, `n_part_left`, `n_part_right`. Series of `MPI_Sendrecv` are suited to a deadlock-free implementation of the above described communication pattern. Portions of the array `field` have now to be exchanged between neighbor processes, along with the elements of the array `x_part` related to the particles residing in the corresponding cells. This is done in subroutine `send_receive_cells` by means of `MPI_Send` and `MPI_Recv` calls. The elements of the spatial-grid array to be sent are copied in suited buffers, and the remaining elements are shifted, if needed, in order to be able to receive the new elements or to fill possibly occurring holes. After sending and/or receiving the buffers to/from the neighbor processes, the array `field` comes out to be densely filled in the range `1:n_cell_new`. Analogously, the elements of `x_part` corresponding to particles to be transferred are identified on the basis of the labelling procedure performed in subroutine `load_balancing` and copied into auxiliary buffers; the residual array is then compacted in order to avoid the presence of "holes" in the particle-index

space. Buffers sent by the neighbor processes can then be stored in the higher-index part of the `x_part` (remember that such an array is overdimensioned). After rearranging the subdomain, subroutine `pushing` is executed, producing the new particle coordinates, `x_part`. Particles whose new position falls outside the original subdomain have to be transferred to a different process. This is done by subroutine `transfer_particles`. First, particles to be transferred are identified, and the corresponding elements of `x_part` are copied into an auxiliary buffer, ordered by the destination process; the remaining elements of `x_part` are compacted in order to fill holes. Each process sends to the other processes the corresponding chunks of the auxiliary buffer, and receives the new-particle coordinates in the higher-index portion of the array `x_part`. This is a typical all-to-all communication; the fact that the chunk size is different for each destination process makes the `MPI_Alltoallv` call the tool of choice. Finally, after reallocating the array `pressure`, subroutine `compute_pressure` is called. Pressure values at the boundary of the subdomain are then corrected by exchanging the locally-computed value with the neighbor process (subroutine `correct_pressure`), by means of `MPI_Send` and `MPI_Recv` calls. The true value is obtained by adding the two partial values. The array `pressure` can now be yielded to the subroutine `field_solver` for the next time iteration.

5 Conclusions

We presented a collective communication service for mobile agents system. The mobility feature allows to dynamically optimize the dispatching of messages by migrating the service itself or, if it is possible, the communication parties. A first implementation in the Jade platform supporting limited communication primitives has been presented. We still lack to provide optimized implementation of messaging forwarding. Also strategies to move the provider to the best nodes and federation of multiple providers need to be implemented. We described how this service is going to be exploited for interfacing heterogeneous parallel applications which interact by collective communication primitives such as the ones defined in the MPI standard. We have finally introduced a real parallel application, used in our ongoing work, which has being used to test our proposed architecture for collective communication service for mobile agents. Due to lack of time we are not able to provide stable results, but performance analysis will be discussed on the conference event.

References

1. F. Rana and L. Moreau: Issues in Building Agent-Based Computational Grids.O. UK Multi-Agent Systems Workshop, Oxford, December 2000
2. H. Tianfield and R. Unland: Towards self-organization in multi-agent systems and Grid computing, Multiagent and Grid Systems Journal, IOS Pres , Volume 1, Number 2 / 2005 89 - 95

3. Z. Li and M. Parashar, Rudder: An agent-based infrastructure for autonomic composition of Grid applications, Multiagent and Grid Systems Journal, IOS Pres, Volume 1, Number 3 / 2005 Pages: 183 - 195
4. R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque: Mobile Agent Programming for Clusters with Parallel Skeletons. VECPAR'2002. Lecture Notes in computer Science, vol. 2565, pp. 614-627, Springer- Verlag, 2003. (ISBN 3-540-00852-7)
5. A. Grama, V. Kumar and A. Sameh: Scalable parallel formulations of the Barnes-Hut method for n-body simulations. Parallel Computing 24, No. 5-6 (1998) 797-822.
6. Message Passing Interface Forum, MPI: A message-passing interface standard. International Journal of Supercomputer Application, 8((3/4)): 165-416, 1994.
7. S. Mitchev and V. Getov. Towards portable message passing in Java: Binding MPI. In Recent Advance in PVM and MPI, col. 1332 of Lecture Notes in Computer Science. Springer Werlag, 1997.
8. B. Carpenter, V. Getov, G. Judd et Al.: MPJ: MPI like Message Passing for Java., Concurrency: Practice and Experience, 12(11):1019–1038, 2000.
9. R. Silva, D. Picinin, M, Barreto et Al.: Performance Analysis of DECK Collective Communication Service.
10. A. Grama, V. Kumar, and A. Sameh.: Foundation for intelligent physical agents. *www.http://www.fipa.org*, 2000.
11. D. Lange and M. Oshima: Programming and Deploying Java Mobile Agents with Aglets. Addison Wesley, 1998.
12. R. Aversa, B. Di Martino, N. Mazzocca, M. Rak, S. Venticinque: Integration of Mobile Agents and OpenMP for programming clusters of Shared Memory Processors: a case study. In proc. of EWOMP (European Workshop on OpenMP), 2001, 8-12 Sept. 2001, Barcelona, Spain.
13. R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque: Mobile Agents for Distribute and Dynamically Balanced Optimization Applications. On High-Performance Computing and Networking (Lecture Notes in Computer Science vol.2110), ed. By B. Hertzberger et al. (Springer, Berlin, 2001).
14. R. Aversa, B. Di Martino, N. Mazzocca and S. Venticinque: MAGDA: A Mobile Agent based Grid Architecture. In Journal of Grid Computing, Springer Netherlands, 2006
15. Foster, I. (2001). The anatomy of the Grid: Enabling scalable virtual organizations. *Lecture Notes in Computer Science 2150*
16. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. N. Karonis, B. Toonen, and I. Foster. Journal of Parallel and Distributed Computing, Vol. 63, No. 5, pp. 551-563, May 2003.
17. B. Di Martino, S. Venticinque, S. Briguglio, G. Fogaccia, G. Vlad: A Grid based distributed simulation of Plasma Turbulence. In: L.T. Yang and M. Guo (Eds.), High Performance Computing: Paradigm and Infrastructure, Wiley eds., 2004
18. Birdsall, C.K., Langdon, A.B.: Plasma Physics via Computer Simulation. (McGraw-Hill, New York, 1985).
19. Di Martino, B., Briguglio, S., Vlad, G., Sguazzero, P.: Parallel PIC Plasma Simulation through Particle Decomposition Techniques. Parallel Computing **27**, n. 3, (2001) 295–314.
20. Fox, G.C., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., Walker, D.: Solving Problems on Concurrent Processors (Prentice Hall, Englewood Cliffs, New Jersey, 1988).
21. Ferraro, R.D., Liewer, P., Decyk, V.K.: Dynamic Load Balancing for a 2D Concurrent Plasma PIC Code, J. Comput. Phys. **109**, (1993) 329–341.
22. Cybenko, G.: Dynamic Load Balancing for Distributed Memory Multiprocessors. J. Parallel and Distributed Comput., **7**, (1989) 279–391.

A Static Parallel Multifrontal Solver for Finite Element Meshes*

Alberto Bertoldo, Mauro Bianco, and Geppino Pucci

Department of Information Engineering, University of Padova, Padova, Italy
{cyberto, bianco1, geppo}@dei.unipd.it

Abstract. We present a static parallel implementation of the multifrontal method to solve unsymmetric sparse linear systems on distributed-memory architectures. We target Finite Element (FE) applications where numerical pivoting can be avoided, since an implicit minimum-degree ordering based on the FE mesh topology suffices to achieve numerical stability. Our strategy is static in the sense that work distribution and communication patterns are determined in a preprocessing phase preceding the actual numerical computation. To balance the load among the processors, we devise a simple model-driven partitioning strategy to precompute a high-quality balancing for a large family of structured meshes. The resulting approach is proved to be considerably more efficient than the strategies implemented by MUMPS and SuperLU_DIST, two state-of-the-art parallel multifrontal solvers.

1 Introduction

Finite Element (FE) applications typically rely on the numerical solution of systems of Partial Differential Equations (PDEs) modeling the behavior of some physical system of interest [16]. From a computational point of view, solving these PDEs involves the solution of large, sparse linear systems whose sparsity pattern depends on the topology of the FE mesh. Since the physical phenomena under simulation may be non-linear and evolving through time, a given FE mesh may require the solution of many linear systems with the same sparsity pattern but with different numerical values. In turn, each of these linear systems can be solved through iterative or direct methods [10]. The use of direct methods becomes particularly desirable if the FE application is such that numerical pivoting is unnecessary and the mesh topology remains unchanged over many iterations. Under this common scenario, successive solutions of (numerically) different linear systems can share the same computation schedule, hence heavy preprocessing, whose cost is amortized over the iterations, can be used for optimization purposes.

In what follows, we regard an FE mesh as a graph with N vertices, representing degrees of freedom (*unknowns*) of the physical system, and edges connecting

* Support for the authors was provided in part by MIUR of Italy under Project *ALGONEXT* and by the European Union under the FP6-IST/IP Project *AEOLUS*.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 734–746, 2006.
© Springer-Verlag Berlin Heidelberg 2006

any two unknowns interacting within some constraint. Under this view, an *element* becomes a fully-connected subgraph (*clique*) of M vertices, where M is a small constant (usually less than 100). Each clique is connected to other cliques by *boundary vertices* in a sparse pattern. The $N \times N$ linear system $\mathbf{Ax} = \mathbf{b}$ associated to the FE mesh at some iteration is *assembled* by computing $\mathbf{A} = \sum_{e=1}^{n_e} \mathbf{A}^e$, and $\mathbf{b} = \sum_{e=1}^{n_e} \mathbf{b}^e$, where n_e is the number of elements of the FE mesh, and the entries a_{ij}^e may be nonzero only when i and j are indices of vertices of the e-th element of the mesh.

Direct methods solve the assembled linear system by decomposing $\mathbf{A}$ into easy-to-solve factors, the most effective strategies for unsymmetric matrices being based on LU decomposition. The elimination order aims at reducing the emergence of *fill-ins* to preserve sparsity, and can be represented by an *Assembly Tree* (AT) whose nodes relate to the elimination of a set of unknowns with the same sparsity pattern. An effective implementation of this idea is the *multifrontal method* [11,15], which can be regarded as a post-order visit of the AT where eliminations at a node employ dense matrix kernels for improved performance [4]. If the linear system arises from an FE application, then the resulting AT can be directly related to the topology of the FE mesh, namely, the former represents a *hierarchical decomposition* of the latter into connected regions, with each AT node corresponding to one such region.

The multifrontal method interleaves phases of *assembly* of larger and larger portions of the FE mesh with *elimination* phases, where a partial LU decomposition is executed on *Fully-Summed* (FS) rows and columns, which are those that need never be updated by further phases. Visiting a leaf of the AT involves computing the matrix $\mathbf{A}^e$ and the right hand side vector $\mathbf{b}^e$ of the related mesh element. Visiting an internal node of the AT entails *merging* the two regions corresponding to its sons and then eliminating the resulting FS rows and columns[1].

Any efficient implementation of the multifrontal method maintains a compact representation of the matrices associated with the nodes of the AT into *dense* two-dimensional arrays storing the non-zero entries of the corresponding matrices. When two regions are merged together into a larger region associated with node n of the AT, during the assembly phase we build an $f \times f$ matrix $\mathbf{A}_n$. If s rows and columns of $\mathbf{A}_n$ become FS due to the assembly, the entries of $\mathbf{A}_n$ can be arranged as

$$\mathbf{A}_n = \begin{bmatrix} \mathbf{S}_n & \mathbf{R}_n \\ \mathbf{C}_n & \mathbf{N}_n \end{bmatrix}, \quad \text{where } \begin{cases} \mathbf{S}_n \in I\!R^{s \times s}, \mathbf{R}_n \in I\!R^{s \times (f-s)}, \\ \mathbf{C}_n \in I\!R^{(f-s) \times s}, \mathbf{N}_n \in I\!R^{(f-s) \times (f-s)} \end{cases} . \quad (1)$$

Submatrices $\mathbf{S}_n$, $\mathbf{R}_n$, and $\mathbf{C}_n$ are called the *FS blocks*. After each assembly, the elimination phase computes the following matrices from $\mathbf{A}_n$:

1. $\mathbf{L}_n \mathbf{U}_n \leftarrow \mathbf{S}_n$;
2. $\bar{\mathbf{U}}_n \leftarrow (\mathbf{L}_n)^{-1} \mathbf{R}_n$;
3. $\bar{\mathbf{L}}_n \leftarrow \mathbf{C}_n (\mathbf{U}_n)^{-1}$;
4. $\bar{\mathbf{A}}_n \leftarrow \mathbf{N}_n - \bar{\mathbf{L}}_n \bar{\mathbf{U}}_n$.

[1] To avoid ambiguity, we will use the term "vertices" exclusively for the vertices of the FE mesh, while the word "node" will be reserved for the vertices of the AT.

After the elimination phase, matrices $\mathbf{L}_n$, $\mathbf{U}_n$, $\bar{\mathbf{U}}_n$, and $\bar{\mathbf{L}}_n$ can be stored elsewhere in view of the final forward and backward substitution activities needed to obtain the final solution, whereas the Shür complement $\bar{\mathbf{A}}_n$ will contribute to the assembly phase associated with the parent of n. Steps 2, 3, and 4 can employ high-performance BLAS Level 3 routines [9], whereas any efficient LU decomposition algorithm can be used in Step 1. The pair of assembly and elimination phases for a region n will be referred to as the *processing* of that region.

This paper describes an efficient parallel implementation of the multifrontal method, whose main feature is a *static, model-driven* approach to work distribution and load balancing among the processing elements. By *static*, we mean that work distribution and communication patterns do not depend on the numerical characteristics of the solution process. This is possible whenever numerical pivoting can be avoided, since a simple *implicit minimum-degree* pivoting strategy ensures the same numerical stability [5]. The benefit of a static strategy is twofold. First, inter-processor communication patterns can be precomputed and optimized; second, heavy preprocessing can be performed to gather data needed to speed up the subsequent numerical computation with negligible memory overhead. Preprocessing time can be amortized over the repeated solutions of systems with the same sparsity structure.

The last few years have seen the emergence of a number of multifrontal solvers (see [2] and references therein). However, only two prominent solvers, MUMPS [1,3] and SuperLU_DIST [8,14], work in parallel on distributed-memory architectures and are capable of solving general unsymmetric linear systems. For this reason, we will compare the performance of our newly developed solver solely with MUMPS, and SuperLU_DIST[2]. MUMPS implements a multifrontal algorithm based on a parallel hierarchical LU decomposition approach similar to the one used in our solver, but it follows a *dynamic* approach to distribute the work between the computing processors. In contrast, SuperLU_DIST starts from the full matrix of the system and partitions it by means of a different technique (based on the identification of *supernodes*).

The results presented in this paper substantiate the claim that the fully static approach adopted by our solver may ensure considerable performance gains over the more complex dynamic solution, provided that simple and effective static load balancing techniques can be afforded by the application domain under consideration.

The rest of the paper is organized as follows. In Section 2 we describe the basic features of a general framework for a parallel multifrontal solver, and introduce the notation used throughout the paper. In Section 3 we provide the details of our parallel multifrontal algorithm. In Section 4 we focus on the model-driven partitioning algorithm devised and present a number of issues related to the AT topology. In Section 5 we compare the performance of our application with MUMPS and SuperLU_DIST on a number of benchmark FE meshes modeling the behavior of porous media. Finally, Section 6 reports some concluding remarks.

[2] In fact, a previous study [2] has proved the superiority of MUMPS over SuperLU_DIST. We decided to retain SuperLU_DIST in our comparison mainly for completeness.

2 A General Framework for a Parallel Multifrontal Solver

In a parallel environment, the work attached to each node of the AT can be speeded up by distributing this work among the available processors. Define a function Π that maps each AT node n into a subset $\Pi_n \stackrel{\text{def}}{=} \Pi(n)$ of available processors that will perform the work attached to that node. Similarly, let Π_n^l and Π_n^r denote the processor sets assigned, respectively, to the left and right son of an internal node n. We say that two arbitrary processors in the same set (either Π_n^l or Π_n^r) are on the *same side*, while a processor in Π_n^l is said to be on the *other side* from a processor in Π_n^r. The pair (Π, AT) defines a *static* processor allocation. In order to carry out the partial LU decomposition associated with node n, the processors in Π_n must first obtain and assemble the Schür complements $\bar{\mathbf{A}}_x$ and $\bar{\mathbf{A}}_y$ previously computed by the processors in Π_n^l and Π_n^r at nodes x and y, children of n. Within this framework, a parallel multifrontal algorithm defined over (Π, AT) must determine for each node n: 1) suitable communication patterns among the processors in the subsets Π_n, Π_n^l, and Π_n^r and 2) how the processors in Π_n cooperate to decompose the newly assembled matrix.

This general framework can be simplified by making some reasonable assumptions on the pair (Π, AT). First of all, it is natural to assume that the number of mesh elements n_e (hence, the number of leaves of the AT) is greater than the number of available computing processors n_p. Therefore, we can find a set of n_p disjoint subtrees that cover all the leaves of the AT and associate each of these subtrees with a single distinct processor. After an initial data distribution, computation on these subtrees can proceed in parallel without any communication involved. We call each of these subtrees a *Private Assembly Tree* (PAT). The computation on each PAT proceeds as in the sequential case amply described in [4]. The (uncovered) subtree of the AT that has the roots of the private subtrees at its leaves is called *Cooperative Assembly Tree* (CAT) since it involves explicit communication between processors. In order to maximize parallelism while limiting the communication volume and enhancing submachine locality, we will consider allocation strategies for which $\Pi_n = \Pi_n^l \cup \Pi_n^r$, for each node n of the CAT.

3 Distributed LU Decomposition Algorithm

Our parallel multifrontal strategy essentially computes the same matrices produced by the sequential algorithm. From here on, these matrices will be referred to as *virtual matrices*. In the parallel algorithm a virtual matrix associated with an internal node n of the CAT is distributed among the processors of Π_n, with each such processor working on a *partial (sub)matrix*. Consider the virtual matrix $\mathbf{A}_n$ of Eq. (1). We partition the rows of $\mathbf{C}_n$ and $\mathbf{N}_n$ into $|\Pi_n|$ subsets denoted as $\mathbf{C}_n^p$ and $\mathbf{N}_n^p$, where $p \in \Pi_n$. The task assigned to processor p is to decompose the partial matrix

$$\mathbf{A}_n^p = \begin{bmatrix} \mathbf{S}_n & \mathbf{R}_n \\ \mathbf{C}_n^p & \mathbf{N}_n^p \end{bmatrix} \tag{2}$$

by means of the four-step sequential algorithm described in Section 1.

Observe that *all* processors in Π_n need the *same* decomposition $\mathbf{S}_n \rightarrow \mathbf{L}_n\mathbf{U}_n$ and the solution to the same triangular system $\bar{\mathbf{U}}_n \leftarrow (\mathbf{L}_n)^{-1}\mathbf{R}_n$ in order to solve different triangular systems $\bar{\mathbf{L}}_n^p \leftarrow \mathbf{C}_n^p(\mathbf{U}_n)^{-1}$ and compute different partial Schür complements $\bar{\mathbf{A}}_n^p \leftarrow \mathbf{N}_n^p - \bar{\mathbf{L}}_n^p\bar{\mathbf{U}}_n$. These partial Schür complements form a row partition of the virtual Schür complement $\bar{\mathbf{A}}_n$ among the processors in Π_n, and will be used in the assembly phase of the father of n. In our solver, we choose to replicate the above common computation in each processor rather then having a single processor gather the relevant data, perform the computation, and then scatter the result to the other processors in Π_n. This master/slave scenario is instead the adopted solution in MUMPS[3].

The way we partition the rows of $\mathbf{C}_n$ and $\mathbf{N}_n$ affects the whole parallel algorithm. We now describe one possible solution that reduces the amount of data exchanged during the assembly phase. Let V_n and V_n^p be the sets of mesh vertices related, respectively, to the rows of the virtual matrix $\mathbf{A}_n$ and the rows of the partial matrix $\mathbf{A}_n^p$ assigned to $p \in \Pi_n$. Clearly, $\bigcup_{p\in\Pi_n} V_n^p = V_n$. Our partitioning makes sure that $V_n^p = V_n \cap V_\rho^p$, where ρ is the root of the PAT assigned to processor p, whence we call V_ρ^p the set of *initial vertices* of p. The main drawback of this simple strategy is that this subset keeps shrinking along the path toward the root of the AT, and will eventually become empty due to rows becoming FS, eventually leaving processor p *potentially idle*: we will address this cause of imbalance in Section 3.2. Note that the vertices related to the columns of partial matrices are the same of the corresponding virtual ones. In fact, the choice of partitioning with respect to the rows (rather than the columns) is totally arbitrary. A totally symmetric column-oriented algorithm can be easily obtained by switching the role of rows and columns.

3.1 The Assembly Phase

In order to build its partial matrix $\mathbf{A}_n^p$, p first has to upgrade the rows of the partial Shür complement computed by p itself in the previous elimination phase, to include the entries of the columns held by processors sharing a subset of its initial vertices. Observe that if more than one processor on the same side has an initial vertex v, then the corresponding rows of their partial Schür complements are the same. As a consequence, during the assembly of $\mathbf{A}_n^p$, processor p can exchange data with at most *one* other processor per vertex, and this processor is on the *other* side from p. No communication within the same side is required during this step. When this assembly step is finished, all processors having v as an initial vertex will have the (same) corresponding row in their partial matrix. This first step of the assembly phase is called the *merging* step.

[3] Note that both approaches afford employing optimized parallel routines for the LU decomposition of S_n at the root of the AT, without the large communication overheads which would be required at internal nodes.

After the merging step, in order to finish the assembly, p must still complete the blocks $\mathbf{S}_n$ and $\mathbf{R}_n$ with the FS rows relative to non-initial vertices. Observe that the missing FS rows can be found somewhere within the same side, since FS vertices are shared by both the left and the right components, and the merging step has already upgraded the corresponding rows. As a consequence, during this step each processor can exchange data with at most *one* processor per missing vertex, and one such processor can always be found on the *same* side. No communication with the other side is needed during this step. We call this second assembly step the *distribution* step, which completes the assembly phase for node n.

3.2 Communication Pattern and Load Balancing

In order to find the communication pattern, we define S_n as the set of *shared vertices* among the processors of the left and right son of node n. As we did for virtual matrices, for each processor $p \in \Pi_n$, we define $S_n^p = S_n \cap V_n^p$ to be the subset of vertices of p which are also shared. What processor p needs to receive during the merging step are the rows whose indices are in set S_n^p and arriving from processors within the other side. Analogously, for the distribution step, we define F_n as the set of vertices that become FS when processing node n of the CAT. Clearly, we have that $F_n \subseteq S_n$. For each $p \in \Pi_n$, we define $F_n^p = F_n \cap V_n^p$ to be the subset of vertices of p which become FS. What processor p needs to receive during the distribution step are the rows whose indices are in the set $F_n \setminus F_n^p$ and arriving from processors within the same side of processor p. Since each processor has multiple potential sources for gathering the data needed for the merging and distribution steps, deciding the optimum communication pattern and the amount of data to gather from each processor involves the solution of a computationally hard problem. To deal with such a problem efficiently, we make the reasonable assumption that the latency in setting up a communication is the real bottleneck. We are then left with minimizing the number of processors that each processor needs to contact, which is an instance of a *Minimum Set Cover* (MSC) problem. The well known greedy strategy for MSC [7] can then be employed to compute a communication pattern whose performance is not too distant from the optimal.

When all the vertices in V_ρ^p become FS, processor p becomes potentially idle. Depending on the shape of the region corresponding to the root of the PAT assigned to processor p, this may happen before the last elimination phase. To avoid a waste of computing power, an idle processor may be assigned a certain number of rows belonging to partial matrices of other processors that are still active. If we limit the possible "donors" of rows to Π_n, the resulting load balancing will feature a high degree of locality easing the parallel forward and backward substitution algorithm [12]. The communications required for balancing can be predetermined during the symbolic analysis phase described in the next subsection. Our approach to balancing is model-driven, in the sense that we use the same cost model described in Section 4 for partitioning the mesh to estimate the load of a processor, and adopt a threshold criterion to maintain a convenient computation/communication ratio.

3.3 Speeding Up the Assembly Phase

As mentioned in the introduction, there is no change in the sparsity pattern of the matrices involved in each of the numerous iterations of the multifrontal solver for a given FE problem. Moreover, our target application domains are such that numerical pivoting can be avoided. Therefore, as already done in the sequential application [4], we can spend time on a preprocessing activity that "simulates" the decomposition process without performing the actual numerical decomposition. During this *symbolic analysis* phase, crucial information regarding the decomposition process are gathered, which can then be used to speed up the subsequent numerical computation. Moreover, in the parallel case, our static work distribution and load balancing strategies make it possible to extend symbolic analysis to encompass the optimization of data exchange between processors during the merging and distribution steps. Specifically, a symbolic representation of a communication pattern is precomputed, so that the matrices can be assembled directly from the receive buffers into their final location so to avoid expensive intermediate buffering and reducing cache-inefficient indirect addressing and conditional branches.

The symbolic data used to speed up the numerical merging step are an extension of the γ-functions used to speed up the assembly phase of the sequential algorithm in [4]. These functions implement inverted references, in the sense that they map the indices of each entry of the *destination* buffers to the indices of the corresponding *source* buffers containing the values contributing to the assembly of that entry.

4 A Model-Driven Partitioning Algorithm

When processing an internal CAT node n, processors in Π_n synchronize their work in two different ways: *external* synchronization is required during the merging step, since processors on opposite sides exchange data, while *internal* synchronization takes place during the distribution step between processors exchanging FS rows within the same side. The load balancing strategy described in Section 3.2 aims at reducing internal synchronization, given that all processors in Π_n will be on the same side during the distribution step associated with the parent of node n. External synchronization time at a node depends on the discrepancy between the running times of the processors computing the left and the right subtree of that node. Balancing the *total* running time on these two subtrees is much trickier than balancing the work on a single node, since the total time depends on the overall topology of the AT and on the map between its leaves and the mesh elements. In order to produce an AT topology capable of yielding an adequate global balancing, we have developed a very simple recursive heuristic that simultaneously partitions the FE mesh into nested rectangular regions and determines the working processors for each region (see Fig. 1 for an example).

The mesh partitioning is driven by the cost function $f(n)$ that models the execution time of the block LU decomposition step performed at a node n of

the AT, depending on the block sizes of the matrices allocated to each processor in Π_n. On a fixed AT, based on $f(n)$ we are able to estimate the computing time $t(n)$ to process its subtree rooted at node n. We synthesize $f(n)$ by least square interpolation from the running times of a suite of sequential block LU benchmarks to be run during the installation of the solver library on a target architecture. When n is a node of a CAT, the function deals only with computation time and not with communication.

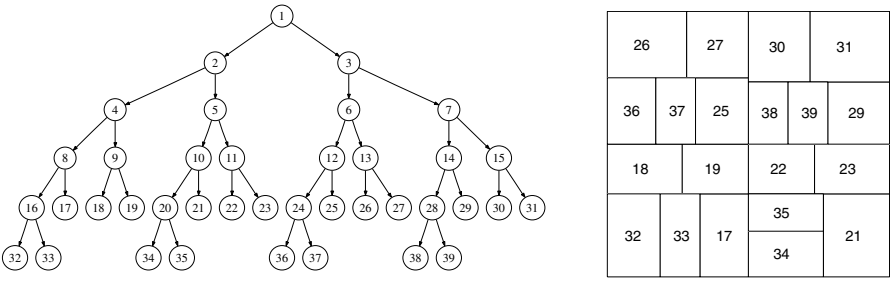


Fig. 1. CAT for 20 processors and the corresponding model-driven partition in 20 regions (one for each leaf) of a square FE mesh. Mesh regions are numbered with the corresponding leaf identifier.

Driven by function $t(n)$, we may determine the region corresponding to each node of the AT by visiting the fixed-shape CAT in a depth-first manner. Starting from the root, which corresponds to the whole mesh, we search for a bisection that minimizes the difference between $t(l)$ and $t(r)$, where $t(l)$ and $t(r)$ are recursively obtained in the same fashion. The resulting partitioning algorithm is exponential in the mesh size but fortunately we can employ simple heuristics to make it affordable. First, we seek nested partitions of rectangular shape, hence the number of possible bisections of a mesh region with m elements is $\Theta(\sqrt{m})$. Second, we limit the exhaustive search activity only at the levels of the CAT whereas we simply decompose each region within a PAT (which will be assigned to single processor) into roughly equally sized subregions.[4] As a result, very few evaluations of the cost function are generally required before finding such a minimum.

After partitioning, we are left with mapping the leaves of the CAT with the processors, trying to enforce as much sub-machine locality as possible. For example, when using a parallel machine made by SMPs with p processors each, every CAT subtree with p leaves should be processed by a single SMP node to avoid slower inter-node communications. To this purpose, our code features a very simple greedy strategy for confining communications between different SMP nodes as close to the root as possible.

[4] This latter simplification affects the quality of the resulting partition minimally, since most of the solver's work is done at the nodes of the CAT.

5 Results

We have compared our solver, dubbed *FEMS (Finite-Element Multifrontal Solver)* against MUMPS v. 4.6 and SuperLU_DIST v. 2.0. In order to quantify the contribution to performance of the model-driven partitioning strategy described in Section 4, we have also set up a modified version of FEMS (called FEMS-M in the following) which uses the METIS package [13] for partitioning. Our target machine is an IBM eServer 575 with 64 Power5 processors working at 1.5 GHz, each capable of a peak performance of 6 Gflop/s. Processors are grouped into 16-processor SMP nodes connected by an IBM High Performance Switch. We used MPI to perform communications among processes, and IBM libraries for sequential and parallel dense linear algebra routines.

Table 1. Test cases main characteristics

Mesh	N. of elements	Matrix order	N. of non-zeros
170×170	28900	436905	33955468
140×140	19600	296805	23028328
400×50	20000	304505	23497308
60×150	9000	137105	10573748

Our test suite comprises rectangular meshes of rectangular elements, modeling scenarios in porous media simulations [6], a computationally challenging application where each finite element has at least 40 degrees of freedom and the involved sparse linear systems are amenable to direct solution without numerical pivoting. For the sake of brevity, we report here the results obtained for four large FE meshes in the suite, whose features are summarized in Table 1. By "large meshes" we mean meshes for which the computation makes an intensive use of main memory. Indeed, as can be seen in Table 2, some execution times are missing due to memory limitations. FEMS, however, is usually able to solve each problem with the smallest number of processors w.r.t. its competitors, which is a clear indication of the fact that the overall memory requirements of FEMS never exceed those of MUMPS and SuperLU_DIST. In fact, the static approach of FEMS allows to compute the amount of memory required by the computation *exactly*, while dynamic approaches generally entail overestimation of memory requirements.

Table 2 summarizes the factorization time of one iteration for various processor configurations, where the lower-order terms due to symbolic analysis, data-distribution, and backward/forward substitution are not included. FEMS and FEMS-M employ implicit minimum-degree ordering, while we let MUMPS automatically choose the fastest pivoting strategy between minimum-degree and METIS ordering, and set up SuperLU_DIST to use minimum-degree ordering on $\mathbf{A}^T + \mathbf{A}$. In order to make a fair comparison, we disabled numerical pivoting in both competitor solvers. Results show that our solver outperforms MUMPS and SuperLU_DIST with both partitioning methods on all test cases and processors

Table 2. Running-time comparison of FEMS, FEMS-M, MUMPS and SuperLU_DIST. Times are in seconds (missing values are due to memory limitations arising when the large test cases are run on a few processors).

		Number of processors								
	Solver	**1**	**4**	**8**	**12**	**16**	**24**	**32**	**48**	**64**
170x170	FEMS	–	12.70	7.14	5.65	4.77	3.99	3.49	3.11	3.01
	FEMS-M	–	16.63	9.20	8.20	6.55	5.38	4.85	4.41	3.98
	MUMPS	–	–	25.21	18.69	18.64	14.79	11.75	8.64	6.68
	SuperLU_DIST	–	–	–	15.29	13.55	21.97	15.12	10.71	11.21
140x140	FEMS	–	7.22	4.15	3.34	2.77	2.38	2.06	1.89	1.80
	FEMS-M	–	9.32	6.89	4.74	4.69	3.82	3.32	3.04	2.84
	MUMPS	–	22.31	15.67	12.24	12.19	10.01	7.21	5.66	4.12
	SuperLU_DIST	–	17.80	11.31	9.57	8.54	8.26	7.60	7.04	7.51
400x50	FEMS	19.18	4.88	2.73	2.03	1.64	1.27	1.06	0.89	0.81
	FEMS-M	–	7.72	5.22	3.85	3.48	3.23	2.51	1.99	1.69
	MUMPS	–	11.82	7.81	5.90	6.86	5.91	4.47	3.84	3.35
	SuperLU_DIST	–	13.86	9.23	8.16	7.45	7.51	6.93	6.59	7.05
60x150	FEMS	8.06	2.08	1.15	0.97	0.77	0.66	0.60	0.53	0.52
	FEMS-M	9.91	3.14	1.71	1.55	1.28	1.16	1.05	0.79	0.78
	MUMPS	16.57	4.78	3.31	2.56	2.87	3.04	2.79	2.70	3.20
	SuperLU_DIST	16.08	6.06	4.01	3.57	3.41	3.26	3.13	2.81	3.18

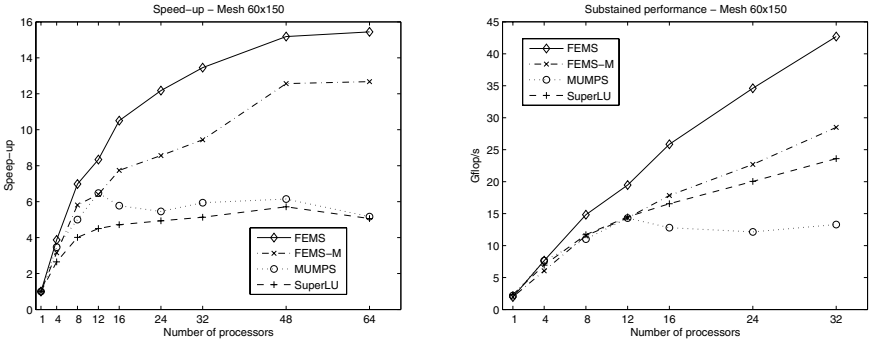


Fig. 2. Parallel performance of the solvers. The graph on the left shows the scalability, whereas that on the right the effective utilization of the computing power. The rate graph is limited to 32 processors due to flop-count limitations of our computing environment.

configurations. The better performance of our solver also in the sequential case shows the benefits of precomputing index functions to speed up the assembly phase. Moreover, it is clear that the model-driven partitioning algorithm substantially improves performance over the use of the general METIS partitioning routines.

Figure 2 gives a better insight into the parallel behavior of the solvers, where we chose to plot the speedup and performance rate graphs relative the 60×150 test case, since the corresponding matrix is small enough to be sequentially factorized by each solver within the available memory. Our solver exhibits considerably higher scalability than the others and makes a better use of the computing power, even if this test case is relatively small. The relatively worse performance of FEMS-M over FEMS mainly depends on a larger external synchronization time since the number of floating-point operations executed by the two versions of our solver is roughly the same.

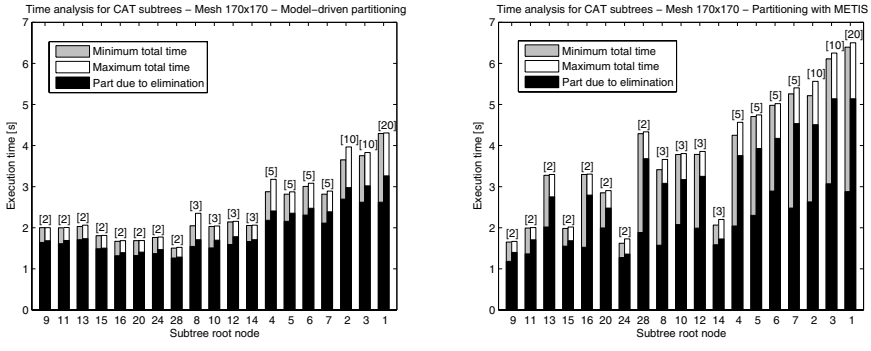


Fig. 3. Differential time analysis of FEMS and FEMS-M on the CAT of Figure 1

A better perspective on the effectiveness of our model-driven partitioning algorithm can be gained by looking at Figure 3 where we compare the running times achieved by FEMS and FEMS-M on each subtree of the CAT shown in Figure 1. Each group of bars represents the total time to compute the subtree rooted at node n (in abscissa), with $|\Pi_n|$ shown above the bars in square brackets; gray and white bars represent, respectively, the minimum and the maximum finishing time of processors in Π_n, with the black portion of each bar representing the fraction of the total time due to the elimination phase. The graph on the left proves that our cost model is very accurate, since the resulting balance of the elimination time is almost perfect. Furthermore, modeling only elimination time seems to be an effective choice also in guaranteeing a good balancing of the overall running time of sibling subtrees. In contrast, observe in the graph on the right that the balancing of the elimination time is much coarser when using METIS, which uniformly partitions the FE mesh into equally sized regions, without considering the distribution of the ensuing computation along the assembly tree.

6 Conclusions and Future Work

We presented a parallel multifrontal linear system solver especially tailored for FE applications, whose main features are a static allocation of work based on the

topology of the FE mesh and a model-driven load balancing technique. Future work will involve the release of a software library providing our FEMS solver. The library will adapt to the computing platform by running a carefully selected suite of microbenchmarks needed to determine the parameters of the cost model used to provide the partitioning. In order to enable to run FEMS also on heterogeneous machines, we are planning to provide the possibility of instantiating the cost model differently on different nodes of the parallel machine. Finally, further research effort will be devoted to the effective application of our model-driven partitioning and static work allocation strategy for unstructured and 3D meshes.

References

1. P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal of Matrix Analysis and Applications*, 23(1):15–41, 2001.
2. P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and X. S. Li. Analysis and comparison of two general sparse solvers for distributed memory computers. *ACM Trans. Math. Softw.*, 27(4):388–421, 2001.
3. P. R. Amestoy, A. Guermouche, J.-Y. L'Excellent, and S. Pralet. Hybrid scheduling for the parallel solution of linear systems. Technical Report RR-5404, INRIA, 2004.
4. A. Bertoldo, M. Bianco, and G. Pucci. A fast multifrontal solver for non-linear multi-physics problems. *International Conference on Computational Science*, pages 614–617, 2004.
5. M. Bianco, G. Bilardi, F. Pesavento, G. Pucci, and B. A. Schrefler. An accurate and efficient frontal solver for fully-coupled hygro-thermo-mechanical problems. *International Conference on Computational Science*, 1:733–742, 2002.
6. M. Bianco, G. Bilardi, F. Pesavento, G. Pucci, and B. A. Schrefler. A frontal solver tuned for fully-coupled non-linear hygro-thermo-mechanical problems. *International Journal for Numerical Methods in Engineering*, 57(13):1801–1818, 2003.
7. T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.
8. J.s W. D., S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.
9. J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, March 1990.
10. J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.
11. I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.
12. A. Gupta and V. Kumar. Parallel algorithms for forward and back substitution in direct solution of sparse linear systems. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing (CDROM)*, page 74, 1995.
13. George Karypis and Vipin Kumar. *METIS: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices Version 4.0*, September 1998.

14. X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110–140, June 2003.
15. J. W. H. Liu. The multifrontal method for sparse matrix solution: theory and practice. *SIAM Rev.*, 34(1):82–109, 1992.
16. O. C. Zienkiewicz and R. L. Taylor. *The finite element method.* Butterworth-Heinemann, fifth edition, 2000.

A Software Architecture Framework for On-Line Option Pricing

Kiran Kola, Amit Chhabra, Ruppa K. Thulasiram*, and Parimala Thulasiraman

Department of Computer Science
The University of Manitoba
Winnipeg, MB, Canada
{kirankkk, amit, tulsi, thulasir}@cs.umanitoba.ca

Abstract. The problem of growing computational complexity in finance industry demands manageable, high-speed and real-time solutions in solving complex mathematical problems such as option pricing. In option trading scenario, determining a fair price for options "any time" and "any-where" has become vital yet difficult computational problem. In this study, we have designed, implemented, and deployed architecture for pricing options on-line using a hand-held device that is J2ME-based Mobile computing-enabled and is assisted by web mining tools. In our architecture, the client is a MIDP user interface, and the back end servlet runs on a standalone server bound to a known port address. In addition, the server uses table-mining techniques to mine real-time data from reliable web sources upon the mobile trader's directive. The server performs all computations required for pricing options since mobile devices have limited battery power, low bandwidth, and low memory. To the best of our knowledge, this is one of the first studies that facilitate the mobile-enabled-trader to compute the price of an option in ubiquitous fashion. This architecture aims at providing the trader with various computational techniques to avail (to provide results from approximate to accurate results) while on-the-go and to make important and effective trading decisions using the results that will ensure higher returns on investments in option trading.

1 Introduction

Option pricing forms a fundamental objective and backbone of financial risk management and decision-making solutions in option trading. Active trading takes place either at the trading floor or through computers with instructions from investors or investment managers. However, once an investor steps away from the workplace, the investor encounters problems of interrupted trading, as the required information is no longer available. In such cases, investors have to rely on the data provided by some source (such as electronic board display). If the investor is away from the building, he/she has to be in continuous touch through other sources such as a broker to get some basic information about the market. However, the information provided by

* Corresponding author.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 747 – 759, 2006.
© Springer-Verlag Berlin Heidelberg 2006

intermediary brokerage firms is generally inadequate especially in the case of computing the option values.

Mobile technology is a new technology that rides a new wave of business innovation. The use of mobile technology for e-business and decision-making strategy is slowly changing the dialogue between investors and traders on the floor of a stock exchange into M-business deals. In this study, we focus on three major issues to achieve ubiquity in derivative markets: (i) mobile commerce aspects in derivative markets (particularly financial options) (ii) various computational techniques used to price options. (iii) mining real-time finance data from web sources. We have incorporated these issues to provide a value-added, ubiquitous service to the trader on the go. We use the terms ubiquitous, and pervasive interchangeably in this paper.

1.1 Motivation for Ubiquitous Pricing

In our preliminary work [1], we have done a feasibility study of derivative pricing using a short-range wireless connectivity with a PDA. Our goal in the current study is to enable wireless trading strategies in ubiquitous fashion and ensure portability for the trader. We have experimented and validated our architecture on J2ME-based mobile emulators, which is applicable for limited and broad range of wireless range networks.

Many researchers attempted to address research issues and possible solutions in mobile commerce. Research efforts by Varshney and Vetter [2] emphasize that mobile financial application is one of the important component of m-commerce, which can replace banks, ATMs, and manual methods by wireless aided services such as online brokerage, and micro payments, etc. The current research is motivated by the concept of providing value-added services in option trading to trader-on-the-move , driven by the principles of operational focus, personalization, multi-channel trading, and handiness[3].

1.2 Vocabulary in Option Trading

An option [4] is a security that gives its owner or holder the right without creating any obligation to trade-in a fixed number of shares of a specified asset (e.g., stocks) at a fixed price (strike price) at any time on or before a specified future (maturity) date.

There are two parties involved in the option trading namely holder and writer. The holder of the option gets the right to buy (call) or sell (put) assets at a predetermined time in the future for a predetermined value; the writer of the option is obliged to deliver (call) or take delivery (put) of the underlying asset.

Two basic styles of options are *European and American*. While the former can be exercised only at maturity, the latter can be exercised at any time prior to maturity. Every option has a set of parameters required to compute the price of the option. Such as the strike price, stock price, risk free interest rate, period of contract, and volatility of the underlying asset. The *strike price* of a call (put) option is the contractual price at which the underlying asset will be purchased (sold) in the event that the option is exercised. The risk-free interest rate (r) is the rate at which an investment (such as simple deposit) would grow without incurring any risk to the capital. The time in

years until the expiration of the option is called *maturity date.* A measure of the change (either up or down movement) of the underlying security over a given period is known as *volatility* (σ). In option markets, accuracy and ability to respond quickly to the fluctuating market is vital for every active investor.

Rest of the paper is organized as follows: We have presented some background details on mobility in derivatives market and option pricing in sections 2 and 3 respectively. We then describe our overall architecture design for real-time option pricing in sections 4 and 5. We discuss a set of results in section 6 and our conclusions in section 7. Our contribution through this work is the design and development of an architecture that enables mobiquitous pricing in the emerging business scenario.

2 Mobile Computing Developments Related to Trading

In this section, after listing possible benefits of ubiquity in trading, we classify the use of the ubiquity into three broad aspects (i) commodity trading (ii) risk management and (iii) services and software. In order to make conscious decisions in an uncertain market place, every investor needs time-critical information in a ubiquitous fashion. Kargupta et al. [5] justify the needs and benefits of reporting time-critical information of stock data through wireless networks. Ajenstat [6] proposed a new idea for automation of on-line derivative (stock options) trading in any place and at any time without the presence of a decision maker. For our research, we integrate time-critical information pervasively. That is, with predefined threshold and boundaries on the price movements of the asset in question, our architecture will initiate new computation in a pervasive fashion whenever "real-time" price of the asset deviates from a predefined value or a predefined range. The "real time" prices on an asset are monitored continuously. For instance, time-critical information for option pricing such as "volatility", and "prime rate" are mined from reliable web sources (presently Yahoo! Finance and Money Cafe). In addition, we have developed our architecture to choose one computational technique (at a time) among various techniques that we have implemented on our back end server. These techniques exploit the time-critical information in order to compute the price of an option accurately for a particular underlying stock.

2.1 Round-the-Clock Trading

There are several factors behind the large dependence on wireless trading. Couple of most compelling factors are: first, wireless devices make this possible to track various market movements anytime or anywhere. Hence. trading activity is becoming a 24-hour-a-day business; second, Roche [7] states that, on black Monday (1987), one option trader lost £55,000 in 5 minutes just for leaving the market for such a short period. The author emphasizes that several market participants would prefer on-line trading while on the move, due to many obvious advantages. Wireless traders do not have the time or power to browse on-line information from hand-held devices and calculate the risk level of a particular asset. However, a trader seeks personalized information to be delivered in a ubiquitous fashion. Currently, investors price options on-line from their desktop computers using various software tools. However, once an investor steps away from the work place, he/she is disconnected from the market

place. There have been some recent advancements to aid traders on the move, SMS (Short Message Service) being one of them. A trader can subscribe to services such as CommSec [8] and Quo Trek [9] in order to be connected on the move. Technically, by subscribing to *equity-alert-services*, one can receive personalized equity information (real-time price information for personalized stock options) over a mobile phone. However, information provided by brokerage firms, via SMS, is insufficient and cannot be used to compute the option price for a particular option contract.

As a first step towards an organized use of mobility devices, Mobility Partner Advisory Council recently announced [10] that "The Chicago Board of Trade (CBOT) is deploying up to 10,000 wireless enabled pocket PC devices in the two years (2004-2005) to floor traders to automate the trading process". CBOT is integrating with Leapfrog technologies to introduce wireless technology for commodity and option trading.

Patsystems [10] has developed software called H-trader, which assists a trader to do Martini Trading. H-trader operates on a mobile phone to trade derivatives across the world. Thinkorswim [11], a brokerage company in Chicago enables the registered trader to perform trading (stock options, and other securities) via web-based, PDA, or Mobile devices.

Windale technologies [12] and FIS-Group [13], present innovative pricing software with various option-pricing techniques that evaluate American and European style call and put options. In addition, support for both desktop environment and Pocket PC versions is available.

Ineffectiveness of these technologies is as follows: (a) the above-mentioned enterprise versions are high-priced products; (b) frequent changes in these products require new updates to the software each time; (c) the computing technique(s) employed for option pricing and their working principles are not explained in their products to the end user; and (d) these products do not present the trader with real-time prices and other parameters (prime interest rate, volatility). Unfortunately, parameters (for example, volatility) used in the computational techniques are highly sensitive to the market fluctuations.

In the current study, we have used various computational techniques as mentioned before. For each technique, our goal is to use real-time information that is mined from reliable web-sources such as Yahoo! Finance. Our architecture uses the cost-effective implementation of a client and server scheme (for instance, Apache Server, and MIDP-interface development are open source environments). In addition, if there are any new updates in computation techniques, we just need to upload the techniques to the back end server rather than uploading them to client. This way, client saves the overhead cost each time when there is a new update. Due to lack of space, we do not describe the algorithm for web mining related to option pricing. We refer the reader to our earlier work [1].

3 Computational Techniques for Option Pricing

Many techniques, such as the binomial tree method, the finite differencing method, and the Monte-Carlo method are being used for pricing options. We describe below

one of the recent computational techniques for pricing options known as finite-difference technique to solve financial models manifested as partial differential equations. We have implemented other computational techniques and for lack of space, we do not describe them here. More information on these and other methods can be obtained from [14][15][16]. We can use any of these techniques as stand-alone modules for option pricing in our architecture described in section 5 and we can incorporate any of the latest pricing techniques in addition to these modules, if necessary. Finite differencing technique is a fundamental numerical approach for pricing financial securities. There are several finite-difference schemes available such as, McCormack scheme, Richardson scheme [17]. Consider the Black-Scholes model [4], a classical option-pricing model

$$\frac{\partial u}{\partial t} + \frac{\sigma^2 S^2}{2}\frac{\partial^2 u}{\partial S^2} + rS\frac{\partial u}{\partial S} - ru = 0 \tag{1}$$

where u is the option price, t is time, σ is volatility, S is the asset price, and r is the interest rate with initial and boundary conditions: $u(0; t) = 0$; and $\lim_{S\to\infty} u(S; t) = S$, $u(S; T) = max(S-E; 0)$ for a call option and $u(S; T) = max(E-S; 0)$ for a put option. We will only consider a call option here. The appropriate discretization for each term in equation (1) is dictated by the individual terms of the PDE together with the required precision and performance constraints. The accuracy of the results can be controlled by the use of a finer grid in the computational time direction as well as the space direction. That is, we iterate the solution process over many computational time steps until we reach a steady state solution. For our research, we have implemented FTCS finite-difference scheme to price options. This is a forward differencing in time direction and central differencing in space direction (for further details on finite-difference schemes please refer to [17][18].

4 Developing Mobile Interface

There are five major platforms available for developing mobile applications such as BREW, Windows Mobile, Symbian, WAP (Wireless Application Protocol) and J2ME (Java 2 Micro Edition). For our research, we developed the architecture on a J2ME platform. J2ME is supported by major carriers (for example Nokia, Motorola) and it is relatively easy to deploy the application for a trader to download and install on mobile devices. We describe in this section J2ME and its implementation details of the current study. J2ME is a stripped down version of Java aimed at machines with limited hardware resources such as a PDA or a mobile phone [19].

We describe details about three important design issues for J2ME that are essential for our study: (i) designing and building a MIDP User Interface (UI) (ii) communication of the MIDP and back end server and (iii) security issues of the network. MIDlet is a MIDP application [20]. Similar to applet, a MIDlet is a managed application. A web browser manages applets, whereas, the Application-Management System (AMS) manages MIDlet. Every MIDlet class handles its own logic and life cycle, which reflects the methods of the MIDlet class. There are three possible methods in a

MIDlet's life-cycle such as startApp(), pauseApp() and destroyApp(). MIDlet enters the active state after the application manager calls startApp(); MIDlet remains in the active state until the application manager calls pauseApp() or destroyApp(). In the pause-App() method, MIDlet is temporarily suspended whereas in destroyApp(), the MIDlet completely terminates the application itself and awaits garbage 15 collection. In MIDP, UI classes are located in the javax.microedition.lcdui package of J2ME. In J2ME, commands are used to create UI objects that behave like buttons (action events in Java); commands such as OK, EXIT, and HELP are characterized by instances of command class.

Option pricing is computationally intensive. Since the required processing power and the memory are both in short supply on mobile devices, computation of option for particular asset is done on the server-end by utilizing the total functionality of Java 2 Standard Edition (J2SE). Moreover, in order to optimize the consumption of resources on mobile devices, it is desirable to keep the communication to a minimum. Therefore, the connection between the server and the device is kept open just long enough to exchange user data.

The connection to dissimilar types of wireless devices will need different forms of connection interfaces. The Generic Connection Framework (GCF) [19] is available in J2ME/CLDC to reflect the need for small-footprint networking for a range of mobile devices. GCF is a hierarchy of interfaces defined in the "javax.microedition.io" package that allows mobile applications readily available to the trader on the network. The GCF interfaces reflect different capabilities and ensure the operations in a logical fashion. MIDP simplifies this GCF to a single connection type called HTTP (Hyper Text Transfer Protocol) and HTTPS (secure HTTP available in MIDP 2). HTTP is built around client requests and server responses, and it has two parts: header and content. The communication format (for example XML, text, and binary) between MIDlets and the back end server in the body of HTTP depends on the design of the application. We tried with GET, HEAD, and POST methods, which are simple to implement and then with XML-RPC and KXML-RPC[1] over HTTP/HTTPS. In our random observations of speed and bandwidth tests, XML tends to have heavy bandwidth between the mobile and the server rather than byte arrays (either it is a string or data of any sort).

In order to provide enough security for data transmission, we used secure HTTP (HTTPS) provided by MIDP 2.0 (If device support MIDP 2.0, it has default HTTPS; for example, Motorola E390 supports MIDP 2.0). On top of that, to provide additional security, we use open source lightweight API called the "Bouncy Castle" library that supports a large number of cryptography algorithms [20]. Therefore, the mobile component of our architecture will be secured. Finally, the task of deployment of the above application (MIDlet Suites) to a specific mobile device can be done using OTA (Over-The-Air installation of MIDlets) or Infrared (IR) or Bluetooth technology. Before illustration of our server design, we will describe the design layout of our architecture with aid of diagrammatic representation.

[1] XML-RPC is a standard way of implementing remote procedure calls (RPC) using XML and HTTP. To accomplish this, it uses XML to mark up all of the method and uses HTTP to transfer the methods

5 Ubiquitous Architecture for Option Pricing

In this architecture (Figure 1), a broker remits a strike price and a contract period of a particular option (with a specific asset) to a subscribed Mobile/PDA/wireless trader. The information provided by the broker is incomplete and can only partially aid in computing the fair value of the option. If the trader (client) needs to decide if entering the option contract is beneficial, the trader has to compute the option value by selecting the *underlying assets* and *computational technique* to be used for computation at the back end server. Moreover, the trader enters the *number of time steps* to be processed for computing the option price. All the above values entered by the trader are sent to the web/compute server for computation.

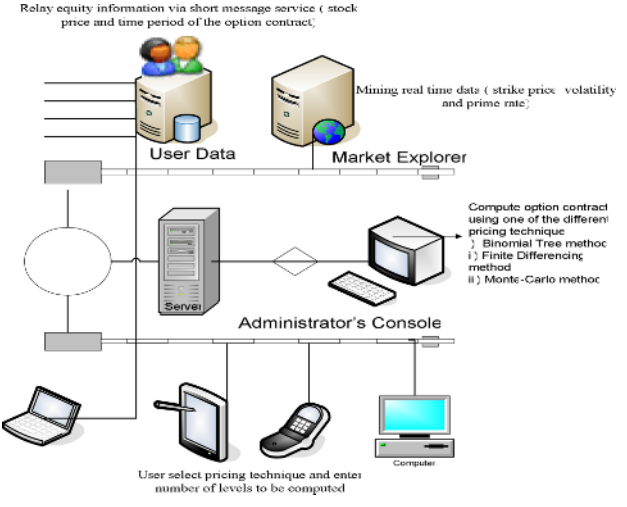


Fig. 1. Mobile Infrastructure

Once the client submits time steps, underlying asset and the computational technique, back end servlet mines (using table mining technique) real-time values such as spot price, volatility, and prime interest rate from the web sources. The server then computes option price with the above real-time values, using finite-differencing technique described in Section 3, or other computational techniques such as binomial lattice or Monte-Carlo method.

Theft and misappropriation are greatest vulnerable factors in wireless trading. Mobile devices can be easily stolen and misused which may result a financial debacle. Consequently, access control and identification of authentic trader are undoubtedly vital in wireless trading. In our framework, transmission mode between client and server are secured in every aspect (as mentioned in section 4). In addition, in order to access his/her portfolio, a trader has to setup and will able to access their accounts that will facilitate customized tables and data analysis based on the underlying computation (for example, to access tables such as risk-free zone, healthy bids, and favored stocks which are discussed in results section). Trader's devices are enabled by

security token and each token generates 5-digit security code that is periodically changed, updated, and acknowledged by the trader to the server or vice-versa.

6 Architecture Results

We have divided our results into two parts: (A) computational results (B) mobile-enabled option trading scenario

(A) Computational Results: In Figure 2, we present one set of results on the computed call option values. We notice that as the strike price increases the call option value decreases, as expected. Implementing the computational techniques in a parallel environment will circumvent the computational cost. This is not the objective of the current study, however.

Intel Corp (INTC) CALL OPTION: Table 1 presents the option values computed at various stock and strike prices for Intel Corp. call option. The real time data for S is 25.52. To make some speculation we have computed the option values based on currently traded stock, and strike prices. This is done to come up with a healthy bid to enter the option contract as presented in table 2.

Table 1. INTC (CALL) option values for varying stock and strike prices

	S= 26.24	S = 25.88	S = 25.52	S = 25.16	S = 25.8	S = 24.54
K=15	11.77553	11.44067	11.1058	10.77094	10.43608	10.19423
K=17.50	9.670142	9.345136	9.0259	8.706682	8.387466	8.156904
K=20	7.720975	7.401749	7.0826	6.763296	6.44406	6.213517
K=22.50	5.945113	5.67445	5.40381	5.133154	4.862501	4.667031
K=25	4.413729	4.143077	3.87243	3.601772	3.383281	3.255619
K=27.50	3.21096	3.034197	2.85743	2.680671	2.503908	2.376246
K=30	2.33158	2.154825	1.979806	1.801299	1.624536	1.502361
K=32.50	1.544512	1.457855	1.371199	1.284542	1.197885	1.134185

Figure 2 shows option values (y-axis) for various strike prices (x-axis). For each strike price there are 6 different stock prices. One (third from the left among six) of them is an on-line value and others are speculated values.

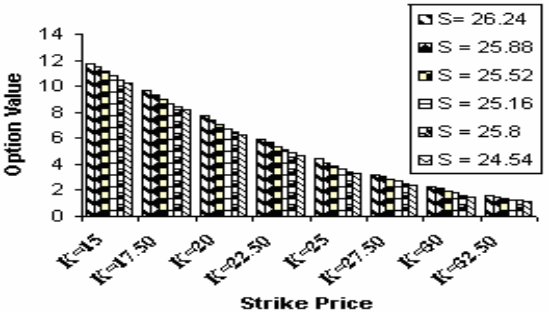


Fig. 2. Option values at various stock and strike prices

Risk-free Zone: To be in the risk-free zone, we have set up a Healthy Bids based on the ASK price and Error rate of online contracts with stock price 25.52. *Ask price* can be defined as the price at which a writer is willing to sell (buy) an asset; also called the offer price. We propose a four-step procedure for calculating "healthy bids" of the option contract: (1) Ask price is mined from online web sources (for example Yahoo!) and option price is calculated (as mentioned in section III) based on real time values. Once we have option price of the contract and online ASK value, we can calculate percentage error, mean error, and finally healthy bid. (2) Calculating percentage error*: Percentage Error rate* can be calculated as: (ASK price – Option price)/ASK price x 100. For our research, we consider errors within the range of (0-10%). If the error rate is more than the specified range, it is discarded. For practical purposes, the real-time error could be improved with advanced computational techniques mentioned earlier. If error is within the specified range, we can continue with the calculation of the *Healthy Bid*. (3) Mean error: Healthy Bid can be calculated based on the mean error rate and is calculated by the formula: abs ((ASK price + option price)/2 – ASK price). (4) If (ASK price – Option Price) < 0: Healthy bid = Ask – mean error and if (ASK price – Option Price) > 0: Healthy bid = option price + mean error As seen in Table1, trader will be provided with various tables with speculated stock values to provide various scenarios of option values and a healthy bid. These tables are made available to the trader (client) from the web/compute server. Depending on the current knowledge of the trader on the behavior of the underlying asset, the trader will be able to select one of the healthy bids (please see table 2 - an example for Intel call option) and instruct the broker to enter the option contract at that bid. If the writer finds this bid comfortable he/she will agree to this price and will agree to sell the underlying asset at the agreed upon strike price at the maturity date. In essence, the investor (client), therefore, has used his/her mobile device ubiquitously to value an option and enter the contract with a level of comfort that the investor can expect a profit from the option contract.

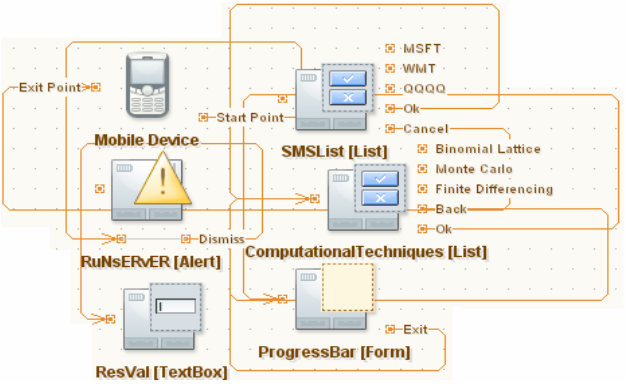
Table 2. Healthy bids for option contract (INTC) when stock is 25.52

Strike Price	ASK-Price	Option Price	Error-R%	Healthy BID
15	11.4	11.1058	2.50%	11.25.29
17.5	9.7	9.0259	6.94%	9.3559
20	7.9	7.0826	10.34%	7.4913
22.5	6.2	5.40381	12.84%	5.8125
25	4.8	3.87243	19.00%	4.33613
27.5	3.6	2.85743	20%	3.05099
30	2.6	1.979806	NA	NA
32.5	1.85	1.3711989	NA	NA

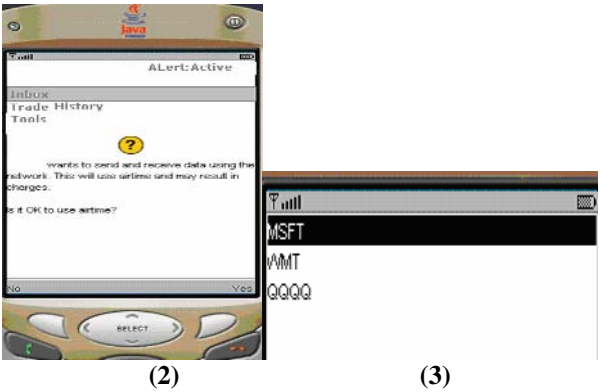
(B) Mobile-enabled Option Trading Scenario

Mobile emulation is done on Net Beans, which is open source software with integration of J2ME Wireless Toolkit platform. Moreover, UEI (Unified Emulator Interface) compatible emulators allow us choosing different devices (for example, NOKIA and

MOTOROLA) from various companies [21]. The following Mobile screens (1-6) describe the flow design and the trading scenario. This architecture enables pricing multiple stock options with various strike prices in ubiquitous fashion. The contract/pricing information of particular stock is updated continuously to the trader on

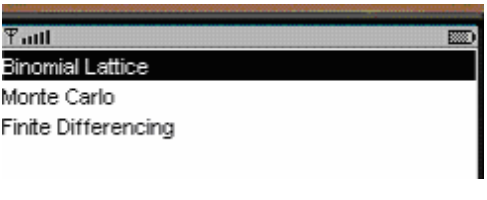


Screen 1. Flow design of mobile terminal and server response

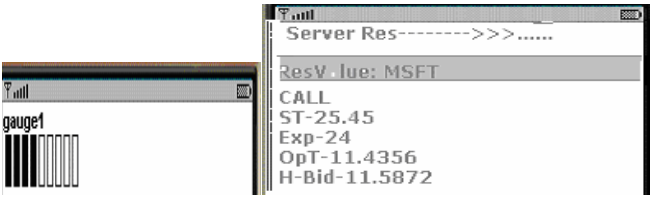


(2) **(3)**

Screen 2 and 3. Alerts from the trading floor-services or brokers; and Active stocks on the trading floor



Screen 4. Details of the contract and computational techniques available for pricing



(5) **(6)**

Screen 5 and 6. Response of the computed results from the web server (back end servlet)

the move by aid of floor-services (exchanges) or brokers in time-to-time fashion. Once the clients get the option price from the web server, he/she will utilize built-in wireless device's small computing power to calculate healthy bids for various stocks in different sectors.

7 Conclusions

Fundamental challenges to design the current ubiquitous software architecture for option trader are from three different scientific domains: option trading, web mining, and mobile computing. Every domain has its own challenges for its functionality. We list the challenges that we faced to build the architecture and our solution to each of the respective challenges and thereby our contribution to the emerging mobiquitous business scenario:

Computation issues: (i) option pricing by itself a computationally intensive problem; (ii) parameters required for computing option price are continuously changing due to market fluctuations. In such a situation, accurate results will depend heavily on appropriate use of current market conditions; (iii) configuring and implementing the existing computational algorithms and handling multiple traders simultaneously and remotely is a challenging task.

Contribution: We have implemented couple of computational techniques (binomial tree method and finite-differencing technique) maintaining accuracy to a large extent. Higher accuracy can be obtained by introducing appropriate new technique (when and if available or developed) in this module of our architecture

Challenges in Web Mining: mining real-time finance data from reliable web sources and forwarding the observed results.

Contribution: Heuristics for single dimensional and two-dimensional tables are employed that are simple yet significant, however not described in the current version due to lack of space.

Challenges in mobile computing: (i) designing and building mobile trading terminal for computing option price any *time* and *any where* is one of the main challenging components of the current research; (ii) as the architecture depends heavily on real time data access, network connectivity and security between mobile client and back end server remain an important issue.

Contribution: To handle these issues we have employed iterative secured-flow-design-approach for building screen logic and layout design of the Mobile interfaces.

In addition, to provide additional security to wireless devices, we employed light-weight API called "Bouncy Castle."

The essence of this framework is in its novelty, which stems from three different domains to enable mobile trader to compute the price of an option in pervasive fashion, which is an important application in option trading. Hence, this is one of the first studies that facilitate the service of option pricing on-the-go. To make the architecture more effective, our plan is to introduce parallel computing of the option pricing problem at hand. Conclusively, our architecture enables an active investor to price options in real-time using various computational techniques in mobiquitous fashion that would pave way for effective investment trading towards higher profitability.

Acknowledgement

The last two authors acknowledge partial financial support from the Natural Sciences and Engineering Research Council (NSERC) Canada through *Discovery Grants* and to the University Research Grants Program (URGP) of the University of Manitoba.

References

1. Kola, K. and Thulasiram, Ruppa K.: UNNATI: Ubiquitous option pricing - a fiNaNce ApplicaTIon framework. Proc. 13th Intl. Conf. on Advanced Computing and Communications, Coimbatore, India, Dec. 2005. (yet to be published)
2. Varshney, U. and Vetter, R.J.: Mobile Commerce:Framework, Applications and Networking Support. MONET, 7, (2002), 3
3. Geoffrey: Mobile Commerce and Wireless Computing Systems. Pearson. Addison Wesley, 1st edition, (2004)
4. Hull, J.C., Options, Futures and Other Derivatives. Prentice Hall, Upper Saddle River, NJ, 6th edition, (2005)
5. Kargupta, H., Park, B., Pittie, S., Liu, L., Kushraj, D., and K. Sarkar: Mobimine: Monitoring the stock market from a PDA. SIGKDD Explorer. News, 3(2) (2002), 37–46
6. Ajenstat, J., and Jones, P.: Virtual decision making for stock market trading as a network of cooperating autonomous agents. In Proc. (CDROM) 37th Annual Hawaii International Conference on System Sciences (HICSS 04), Big Island, Hawaii, Jan. (2004)
7. Roche, J.: Mobile Trading Comes of Age. In United Nations Conference on Trade and Development, September (2002)
8. CommSec: Keep up the market, whereever you are. CommSec, http://www.satisfac.com.au/ convCommSecsharetrading.htm, Sep. (2003)
9. QuoTrek: Real time on-line data. http://www.quotrek.com/default.asp, Feb. (2003)
10. Microsoft: Mobile Solution Partner. Pit trader work wirelessly with pockets pcs and solution from phatware. Leapfrog technologies. In Mobility Partner Council, San Diego, CA, (2003)
11. Thinkorswim Inc: stock option trading: online brokerage services: http://www.thinkorswim.com/tos/displayPage.tos?webpage=softwareLobby. March. (2006)
12. Windale Technologies: Option Pricing Analysis X Software. http://windale.com/optionsx.php3, June (2005)
13. FIS Group: Option calculator. www.fis-group.com, 2003.

14. Barua, S., Thulasiram, Ruppa K. and Thulasiraman, P.: High Performance Computing for a Financial Application using Fast Fourier Transform. Proc. European Parallel Computing Conference, (EuroPar 2005), Lisbon, Portugal, LNCS Vol.3648; Aug.30-Sep2, (2005) 1246-1253
15. Rahmail, S., Shiller, I., and Thulasiram, Ruppa K.,: Different estimators of the underlying asset's volatility and option pricing errors: Parallel Monte- Carlo simulation. Proc. Intl. Conf. on Computational Finance and its Applications (ICCFA), Bologna, Italy, April 21-23, (2004), 121-131
16. Thulasiram, R.K., Litov, L., Nojumi, H., Downing, C. and Gao, G., Multithreaded Algorithms for Pricing a Class of Complex Options. Proceedings (CD-RoM) of the IEEE/ACM Int. Parallel and Distributed Processing Symposium (IPDPS) April (2001)
17. Tannehill, J.C., Anderson, D.A.., and Pletcher, R.H.: Computational Fluid Mechanics And Heat Transfer. Routledge; 2nd edition, April (1997)
18. Thulasiram**,** R.K., Zhen, C., Chhabra, A.., Thulasiraman, P., and Gumel, A.: A second order L_{0} stable algorithm for evaluating European options. Intl. J. High Performance Computing and Networking, (to appear)
19. White, J., An introduction to java 2 micro edition (j2me); java in small things. In ICSE '01: IEEE CS Proceedings of the 23rd International Conference on Software Engineering, Toronto, Ontario, Canada, (2001) 724–725
20. Itani, W., and Kayssi, A.: J2me application-layer end-to-end security form-commerce. Journal of Network and Computer Applications, 27(1) (2004) 13–32.
21. Kaisa, N. Mattila, N.V., and Ruuska, S., Design: Designing mobile phones and communi cators for consumer needs at nokia. Interactions, 6(5) (1999) 23–26.

An Open Source Web Service Based Platform for Heterogeneous Clusters*

F. Almeida[1], S. Barrachina[2], V. Blanco[1], E. Quintana[2], and A. Santos[1]

[1] Dpto. Estadística, .I.O. y Computación
Universidad de La Laguna, Spain
falmeida@ull.es
[2] Dpto. Ingeniería y Ciencia de Computadores
Universidad Jaime I de Castellón, Spain
quintana@icc.uji.es

Abstract. We present our joint effort to develop a web based interface for the GNU Scientific library and its parallelization. The interface has been developed using standard web services technology to enable the use of non local resources to execute parallel programs. The final result is a computing service where sequential and parallel routines demanding high performace computing are supplied. The design allows to incorporate new servers and platforms with a small number of software requirements. We also introduce an open source development environment to allow developers to cooperate in the parallelization of the GNU Scientific library codes. These codes also will be available trough the web based interface to end users. Performance results are shown for some GSL codes in two cluster heterogeneous systems using the interface enabled with web services technology.

1 Introduction

High Performance Computing (HPC) systems can be implemented with standard hardware available in the market. But users use to build these systems step by step. On the other hand, user from different institutions can colaborate to build HPC systems joining their resources. This fact makes that these systems becomes heterogeneous. The access to this computing resources can be facilitated through the use of web technologies. This wellknow technologies can help to carry the HPC resources to users and help to minimize the access cost to this systems by end users.

The use of web services based technologies is becoming very popular for web-based applications. A Web Service (WS) basically consists of the use of open standards to connect applications through a communication network. This approach allows to homogenize the access to those services and eases the development at the clients. Since the information exchange employs *Extensible Markup*

* Authors are listed in alphabetical order. This work has been partially supported by the EC (FEDER) and the Spanish MEC (Plan Nacional de I+D+I, TIN2005-09037-C02).

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 760–771, 2006.
© Springer-Verlag Berlin Heidelberg 2006

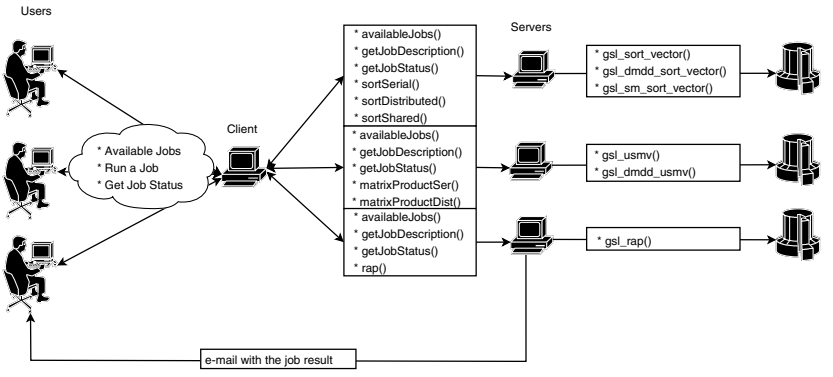


Fig. 1. Generic view of the web service implementation

Language (XML) documents, the application is independent of the transport layer protocols and interfaces.

Standardizing and easing the access to distributed resources is a tendency that has also been observed in the parallel computing community, in projects such as Netsolve [1].

The use of WS technologies enables the access to nonlocal resources, with well-known examples being the execution of parallel programs in parallel or geographically distributed machines [2], or the interaction among heterogeneous devices [3]. Another advantage is that WS communicate through any common firewall security measures without requiring changes to the firewall filtering rules.

In this paper, we describe a parallel computing service that has been implemented according to the W3C (World Wide Web Consortium) recommendations for WS development [4]. The service is addressed to users with little (or even without) experience in programming and facilitates the execution of sequential or parallel routines demanding high performance computing. Flexibility to enlarge the service with new computational proposals is an important issue considered in our design strategy. Furthermore we pursue that, as the service is extended, platforms can be easily incorporated with only a small number software requirements. We focus on the use of the services on a cluster of heterogeneous systems where each node may have different computational capabilities.

It is worth noticing that a WS based design adds, by itself, new facilities to those provided by some other projects [1]. Moreover, a web based interface enables the use of the system by a larger number of users, including those without any knowledge in programming (both sequential or parallel). A second difference with respect to [1], is the use of W3C recommendations for the WS development, that imposes a modular design based in layers (levels). Layers can be independently extended or even replaced to provide new facilities without altering the rest of the parts of the system.

At the back-end, our computational service currently implements (fig. 1) the GNU Scientific Library (GSL) and its parallel version in the Pelican library [5,6]

for heterogeneous systems. Our goal is to enable the use of the routines in both libraries via a web interface as part of a free service to the scientific community. The service has been implemented using the HTTP Apache server and the NuSOAP library [7]. Users registered at the service can execute precompiled routines in the GSL and Pelican libraries providing their own input data. However, since the proposed methodology is generic, the approach is applicable to any other sequential or parallel library. Although we are not using knew concepts, our main design effort has been to connect all these ideas together, and make them to work into a successful tool that will be very useful for many scientific communities.

The paper has been structured as follows. In section 2 we describe the software architecture of our parallel approach for the GNU Scientific Library. Section 3 describes the Open Source development infrastructure to allow other developers to collaborate with the project. Section 4 summarizes some basic web services concepts. In section 5 we present the web service interface. We describe the technology used for the client and for the server, and its use is illustrated through a very simple combinatorial optimization routine. We finalize the paper in section 7 with some concluding remarks and future lines of work.

2 The Parallel GNU Scientific Library (Pelican)

GSL is a collection of hundreds of routines for numerical scientific computations. Although coded in ANSI C, the routines present a modern application programming interface for C programmers, and the library employs an object oriented methodology allowing wrappers to be written for very high-level languages. The library includes numerical routines for complex arithmetic, matrices and vectors, linear algebra, integration, statistics, and optimization, among others. The Pelican project collects the joint efforts towards the parallelization of GSL using MPI and OpenMP. As a result, the Pelican library is portable to a wide range of parallel architectures, including distributed and shared-memory multiprocessors, hybrid systems -consisting of a combination of both types of architectures-, and clusters of heterogeneous nodes. Besides, the Pelican library targets two different classes of users: a programmer with an average knowledge of the C programming language but with no experience in parallel programming, that will be denoted as user A, and a second programmer, or user B, that is familiar MPI or OpenMP.

The Pelican library has been designed as a multilevel software architecture composed of four layers; see Fig. 2. Following a common approach in computer networks, each layer offers certain services to the higher layers and hides those layers from the details on how these services are implemented. The design has been tested and validated [8], [6], [9] and the performance obtained is acceptable when compared with that of similar alternatives. We briefly describe next the contents of the major layers in the software architecture.

The *User Level* (the top level) provides a sequential interface that hides the parallelism to user A and supplies the services through C/C++ functions

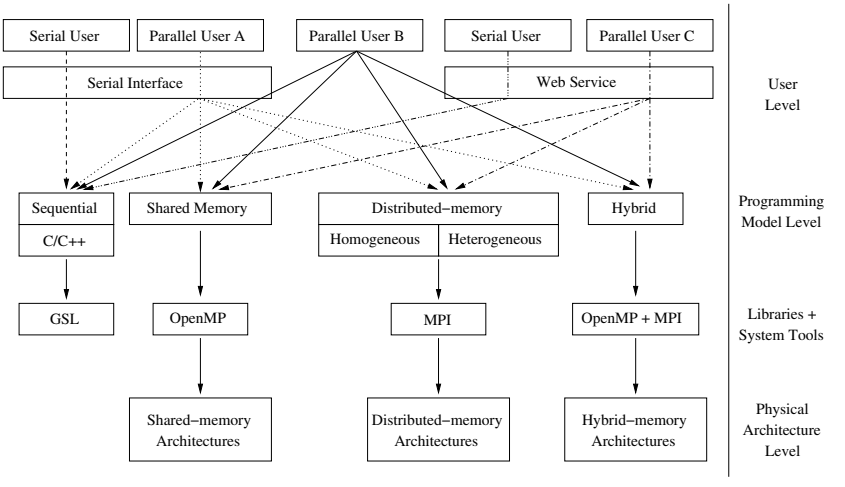


Fig. 2. Software architecture of the Pelican Library extended with the WS

according to the prototypes specified by the (sequential) GSL interface (for example, a `gsl_sort_vector()` routine is provided to sort a `gsl_vector` data array, possibly in parallel).

The *Programming Model Level* provides a different instantiation of the GSL library for each one of the computational models: sequential, distributed-memory considering both homogeneous and heterogeneous systems, shared-memory, and hybrid. The semantics of the functions in the Programming Model Level are those of the parallel case so that user B can invoke them directly from her own parallel programs. The function prototypes and data types in user A codes are mapped into the appropriate ones of this level by just a renaming procedure at compilation time. The Programming Model Level implements the services for the upper level using standard libraries and parallelizing tools like (the sequential) GSL, MPI, and OpenMP.

At the *Physical Architecture Level*, the design includes shared-memory platforms, distributed-memory architectures, and hybrid and heterogeneous systems (clusters of nodes with shared-memory and processors with different capabilities). We map one process per processor of the target parallel system where, in order to balance the computational load, a process will carry out an amount of work that is proportional to the performance of the corresponding processor. The performance of the parallel routines will depend on the adequacy between the programming paradigm chosen by the user and the target architecture.

In this paper we introduce a new module in the Pelican software architecture, as depicted in Fig. 2. This module is located at the user level and comprises a WS application that provides the sequential and parallel routines as a free parallel computational service accessed through a common web interface. In the scheme, the user supplies the input data for the routines, and the servers execute the sequential or the parallel routines, according to the user's interfaces.

3 Source Code Liberation

3.1 Motivation

The number of routines in the GSL library roughly approximates to one thousand, therefore, the whole parallelization of the library would be a very long time project for a small group of people. We have implemented the necessary infrastructure to expand the project as a true open source one, allowing people outside the project to cooperate in it. We are currently publishing the developed code under the GNU General Public License (GPL), as stipulated by the original source.

The fundamental goal to develop this infrastructure is to increment the number of codes to be parallelized. By providing free access to the repository we allow the potential future developers to get source code samples, and also future developments can be set at a centraliced fixed location.

Currently we provide the following services: A Control System Version managed by *Subversion*. Mailing lists to discuss topics related to the project. Free access for stable releases. Documentation. Access to the web service from the same portal.

3.2 Version Control System

Our current version control system is supported in *Subversion*. *Subversion* is a control version application quite similar to *CVS*. Although we analyzed several alternatives like (*CVS*, *Subversion*, *Darcs*, *GNU Arch* and *Bazaar-NG*), we finally decided to use *Subversion* for the following reasons:

- Is a free software application.
- Is becoming quite popular and widely used.
- Is a centralized system.
- Can be integrated with Apache and therefore can be accessed from the 80/TCP port.
- Improves the main drawbacks found in *CVS*.
- Provides a client for most of current the operating systems.

Some of the former are requirements are imposed by the target environment platform where the project is located. To be more precise, it is a well known fact that this kind of projects must coexist with the security policies of the organizations. In particular, at La Laguna University, as in many organizations, most of the communication ports are closed. The access to the service must be devised through common ports as the ports 22 (SSH) or 80 (HTTP). We discarded CVS since it forced us to create an account on the server for every user, and the use of *Subversion* integrated in Apache seems to be a suitable option.

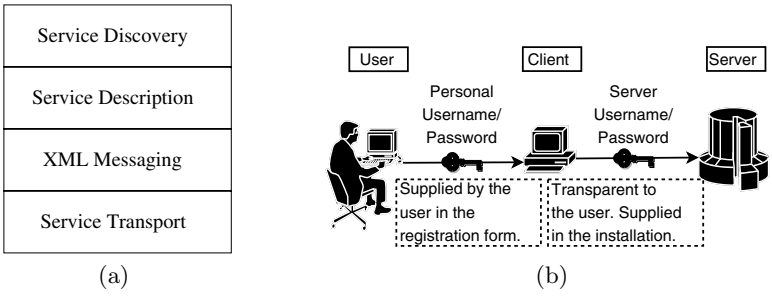


Fig. 3. The web services protocol stack and security implementation

4 Web Service Concepts

A WS is usually regarded as a client/server application where the interaction between those is implemented using open standards. The services are commonly associated to Internet but this is not strictly necessary. Examples of WS are the Google API [10] or the Amazon WS [11].

WS are typically described using a protocol stack similar to that shown in Fig. 3(a). In particular, the WS protocol stack specifies how a WS is described, discovered, and implemented.

1. Transport: The messages generated on a WS are independent of the transport layer. Standard, well-known protocols, as HTTP, SMTP, or FTP, are commonly used.
2. Messages: The information is exchanged in XML documents. Protocols like SOAP (*Simple Object Access Protocol*) [12] or XML-RPC (XML-Remote Procedure Call) [13] can be used for that purpose.
3. Description: This layer describes the *operations* (specifically, routines and their input/output parameters) available at the WS. The description follows a standard known as WSDL (*WS Description Language*) [14].
4. Discovery: At this level the catalog of available WS, known as UDDI (*Universal Description, Discovery, and Integration*) [15], is maintained. The aim is to facilitate the search and publication of services.

An important issue on the web service standard are the security matters, for example, there is a communications protocol WS-Security (Web Services Security [16]) that provides means for applying security to Web Services. Some other security measures to be made are related to the message encryption or the client authentication. For simplicity reasons, we decided to use HTTPS to encrypt the message and the facilities provided by Apache to protect the access to a resource with a username and a password to authenticate clients (fig. 3(b)). The access will be impossible at all if the pair (username, password) is unknown.

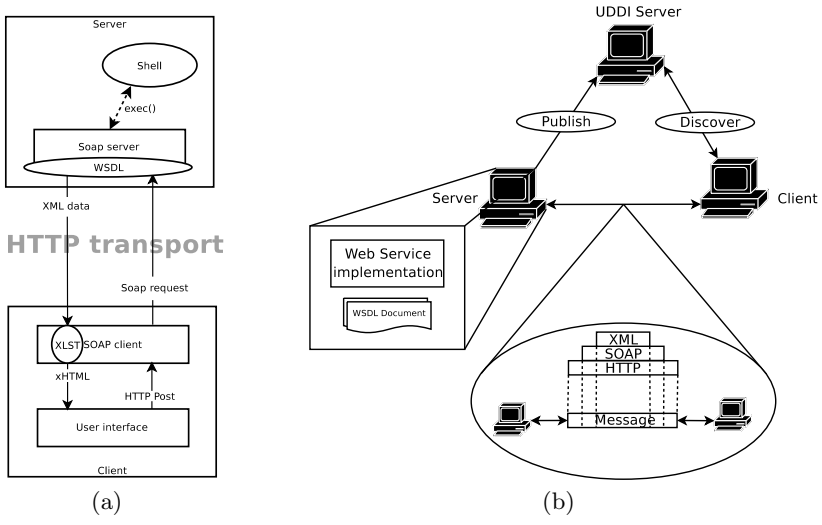


Fig. 4. The Pelican WS modules and WS stack

5 The Pelican WS Architecture

Our WS enables the execution of parallel tasks on all the machines registered at the service. Although the current interface employs web pages, interfaces based in protocols as SMTP (e-mail) or FTP could also be easily incorporated. We have installed the system in two testbed clusters, one at the University of Jaume I at Castellón and a second one at the University of La Laguna, with the following characteristics:

- `ra.act.uji.es`: Cluster of symmetric multiprocessors (hybrid system) composed of 34 nodes with dual Intel Xeon processors connected via a Myrinet network.
- `tegasaste.pcg.ull.es`: Cluster of symmetric multiprocessors (hybrid system) composed of 24 nodes with dual Intel Xeon processors connected via a Gigabit and Infiniband networks.

Two modules compose the WS package, namely, the server (side) module and the client (side) module; see Fig. 4(a). The modules implement the three lower level layers of the WS stack: Service Description, XML Messaging, and Service Transport. The Service Discovery level has not been implemented for security reasons. Thus, system managers still control the client services accessing the clusters via traditional techniques for authentication.

5.1 The Technology

Several approaches are currently available to implement servers as WS: ISS + Microsoft .NET, Apache + NuSOPA or Tomcat + Axis. Although .NET is a

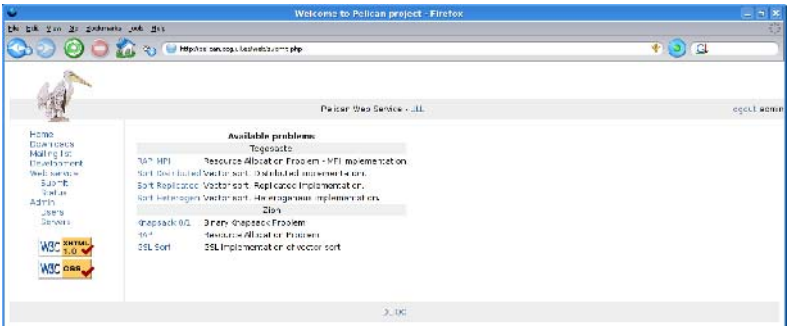


Fig. 5. List of available routines accessible through the WS

software approach that is being widely used under specific operating systems, the proprietary nature of this product leaves it out of the scope of our project, which attempts to be portable to as many platforms as possible. Tomcat and Axis are Java-based solutions that agree with the portability goal of our project and so it does Apache. In the end, we decided to use the combination of Apache+NuSOAP due to their low resource requirements. Figure 4(b) shows the web services stack in our implemention.

5.2 The Client

The client provides a user interface and translates the requests into queries for the server.

The users of the service should be registered at the client. This registration is made at the client web page through a form. The information required consists of an e-mail address, to collect the results and news, the username and password to be authenticated at the client and some information relative to her organization to help the system manager to decide if the registration succeeds or not. Once these information has been submitted the account will not be activated until the organization, through a privileged user (the system manager), decides whether the request is accepted or not. At this moment the user is notified about the resolution.

The current implementation consists of a web interface that the user can employ to access lists of several routines of the GSL and Pelican libraries; see Fig. 5.

Each entry of the lists shows a brief description of the routine. Tasks are grouped according to the servers supporting them so that, when execution of a routine is requested, the target platform is implicitly selected.

We next illustrate the implementation strategy using a dynamic programming routine to solve a Resource Allocation Problem (RAP) [17]. For a specific allocation function, an instance of the RAP is determined with the number of activities and the number of resources to be assigned. An HTML form is dynamically

Fig. 6. Servers management option

generated according to the description of the job to input the parameters needed for its execution. Listing 1.1 shows the HTML form for the RAP example.

The service notifies the user via e-mail when the job is finished, with the standard output of the job incorporated as an attachment to that mail. Also, a web interface allows to learn the state of the jobs submitted to execution , and to collect other results stored by the job into disk files.

As it has been previously mentioned, users with management privileges can be added. These users may access to the management options at the client (fig. 6) and they will be responsible for adding and removing users and servers.

5.3 The Server

The server manages all issues related to the jobs: making these available at the service, and controlling their state and execution. The server needs to know how a job will be executed and how to query the state of a job in execution on the server supporting it. For that purpose, two methods of a PHP class, the class `PelicanJob`, should be overwritten. These methods enable the execution of the job (under the queue system) and to ask for the state of the jobs. In addition, an XML description of each available routine needed to specify the job. As an example, Listing 1.1 shows the XML description for the RAP. The tag <`name`> holds a representative identifier for the routine and the tag <`binary`> is the path to the executable file. Then a description for the problem and the arguments of the routine with their data types are introduced. Once the user submits a job request, the server executes the associated binary code with the arguments supplied by the user. The binary is a precompiled file of the code in Listing 1.2. Thus, new services are easily incorporated to the service just by adding the description XML file with the precompiled code to the server.

Listing 1.1. Describing a Problem.

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="job.xsl"?>

<job xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNameSpaceSchemaLocation="job.xsd">
        <name>RAP</name>
        <service_name>rap</service_name>
        <binary>bin/rap</binary>
        <description>Resource Allocation Problem</description>
        <argument type="integer">
                <name>num_act</name>
                <sdesc>Number of Activities</sdesc>
                <ldesc>The number of activities available to assign
                        resources.</ldesc>
        </argument>
        <argument type="integer">
                <name>num_res</name>
                <sdesc>Number of Resources</sdesc>
                <ldesc>The number of units of resource.</ldesc>
        </argument>
</job>
```

Listing 1.2. RAP GSL Dynamic Programming: main code.

```
int main(int argc, char *argv[]) {
        int N;              /* Activities */
        int M;              /* Number of Resources */
        int result;

        if (argc != 3) {
                printf("Usage:\n");
                printf("\trap <NUM_ACTIVITIES> <NUM_RESOURCES>\n");
                return 1;
        }
        N = atoi(argv[1]);
        M = atoi(argv[2]);
        result = dp_rap_value(&f, N, M);
        printf("# optimal_value = %d\n", result);
        return 0;
}
```

For security reasons, the server can be protected using a username and a password. We have used the HTTP server (Apache) facilities, so that every access to the path where the server was installed will demand the client authentication. The pair username/password will be supplied during installation and it should be delivered to any authorized client.

6 Computational Results

A sorting routine is used next to illustrate the performance of the distributed-memory programming model. For generality, the library chosen the well-known Parallel Sort by Regular Sampling (PSRS) algorithm, introduced in [18]. Figure 7 shows the results of the execution performed using both the heterogeneous and the homogeneous replicated versions of the routine. The heterogeneous version automatically performs a data distribution proportional to the computational capabilities of the processors involved in the computation.

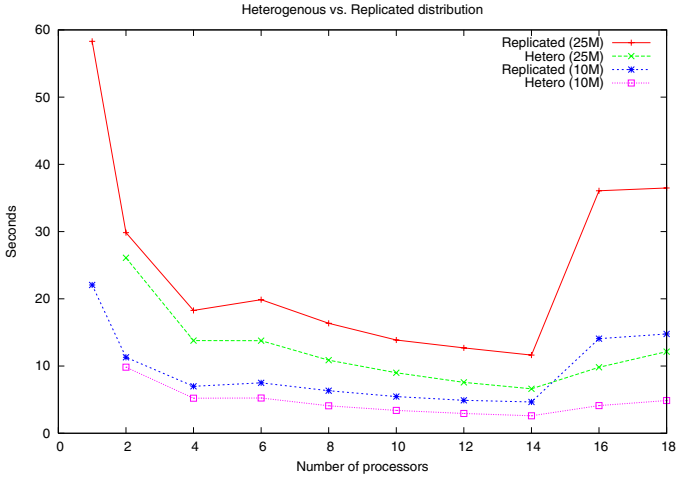


Fig. 7. Sorting in an heterogeneous cluster. Heterogeneous vs homogeneous replicated versions.

We generated double data vectors with entries randomly distributed. Problem sizes vary from 10×10^6 until 25×10^6. The heterogeneous cluster has been obtained from Tegasaste using different type of nodes. Processors are included at the computation according to their computational capabilities, i. e., first the faster processors are considered and later on the slow processors. That means, for instance, that an execution using six processors uses 2 $\times$ nodes Dual Xeon 3.2GHz (4 processors) and 1 $\times$ node Dual Xeon 2.6GHz (2 processors). The running times shown at Fig. 7 do not measure the initial data distribution. The homogeneous replicated version produces the usual decrease in performance and peaks in the running times as a consequence of the introduction of slow processors. On the other hand, the use in the heterogeneous version of a data distribution proportional to the computational power of the processors results in a softer and improved execution time curves.

7 Conclusion and Future Work

We conclude that our aim of developing a web based interface for the GNU Scientific Library and its parallelization has been successfully achieved. The design, based in web services technologies, is quite simple and enables to incorporate new computational resources (clients and servers) with a minimum effort. In order to facilitate the use of the service, we plan to include the possibility of job submission via e-mail. A second addition to the system would consist of letting the system choose the execution platform automatically, depending on an estimation of the one which would provide the smallest response time.

We have set the bases for an Open Source development environment in order to help developers to parallelize new codes from the GSL library and put them

available to end users trough the web service interface. We provide a versioning system (SVN), mailing lists, documentation, public access to stable releases and access to the web service interface through the portal of the project.

References

1. netsolve: NetSolve project web site (2006) http://icl.cs.utk.edu/netsolve.
2. Puppin, D., Tonellotto, N., Laforenza, D.: Using web services to run distributed numerical applications. In: EuroPVM/MPI 2004. Number 3241 in Lecture Notes in Computer Science, Springer-Verlag, Berlin, Heidelberg (2004) 207–214
3. Lioupis, D., Stefanidakis, M.: A web service for embedded distributed computation. In: Euromicro Conference on Parallel, Distributed and Network-based Processing. (2005) 20–25
4. W3C: World wide web consortium (2006) http://www.w3.org/.
5. Galassi, M., Davies, J., Theiler, J., Gough, B., Jungman, G., Booth, M., Rossi, F.: GNU scientific library reference manual. (2002) Ed. 1.2, for GSL Version 1.2.
6. Aliaga, J., Almeida, F., Badia, J., Barrachina, S., Blanco, V., Castillo, M., Dorta, U., Mayo, R., Quintana, G., C.Rodriguez, F.Sande: Parallelization of gsl: Architecture, interfaces, and programming models. In: Proc. of the 11th European PVM/MPI Users' Group Meeting in conjunction with DAPSYS'04 (EuroPVM/MPI 2004). Volume 3241 of Lecture Notes in Computer Science., Budapest, Hungary (2004) 199–206
7. NuSOAP: NuSOAP - soap toolkit for php (2006) http://sourceforge.net/projects/nusoap.
8. J.Aliaga, Almeida, F., Badía, J., Barrachina, S., Blanco, V., Castillo, M., Dorta, U., Mayo, R., Quintana, E., Quintana, G., Rodríguez, C., de Sande, F.: Parallelization of GSL: Performance of case studies. In: Proc. of the Workshop on State of the art in Scientific Computing (PARA'04), Copenhagen, Denmark (2004) 83
9. Aliaga, J., Almeida, F., Badía, J., Barrachina, S., Blanco, V., Castillo, M., Dorta, U., Mayo, R., Quintana, E., Quintana, G., Rodríguez, C., de Sande, F.: Parallelization of the GNU scientific library on heterogeneous systems. In: Proc. of the Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar'04), Cork, Ireland (2004) 338–345
10. Google: Google api (2006) http://www.google.com/apis/.
11. Amazon: Amazon web service (2006) www.amazon.com/gp/aws/landing.html.
12. SOAP: Simple object access protocol (SOAP) (2006) http://www.w3.org/TR/soap/.
13. XML-RPC: Xml-rpc home page (2006) http://www.xmlrpc.com/.
14. WSDL: Web services description language (WSDL) 1.1 (2006) http://www.w3.org/TR/wsdl/.
15. UDDI: Universal description, discovery and integration (UDDI) (2006) http://www.uddi.org/about.html.
16. WS-Security: Web services security (ws-security) specification (2006) www-106.ibm.com/developerworks/webservices.
17. Ibaraki, T., Katoh, N.: Resource Allocation Problems. Algorithmic Approaches. The MIT Press (1988)
18. Li, X., Lu, P., Schaeffer, J., Shillington, J., Wong, P., Shi, H.: On the versatility of parallel sorting by regular sampling. Parallel Computing **19**(10) (1993) 1079–1103

Cost Models and Performance Evaluation of Similarity Search in the HON P2P System

Mouna Kacimi and Kokou Yétongnon

University of Bourgogne, Laboratoire LE2I
Sciences et Techniques, 21078 Dijon Cedex France
{mouna.kacimi, kokou.yetongnon}@u-bourgogne.fr

Abstract. Similarity searching is particularly important in fully distributed networks such as P2P systems in which various routing schemes are used to submit queries to a group of relevant nodes. This paper focuses on the maintenance cost models and performances of similarity search in the HON P2P system, where data and peers are organized in a high dimensional feature space. We show through extensive simulations that HON has a low maintenance cost and is resilient to peers' failures.

1 Introduction

P2P systems and applications are gaining in popularity, spurred by the need for seamless interconnection of services and resources; distributing data indexes and processing among multiple nodes; and sharing large amount of data in dynamic ad hoc environments. Early popular file sharing P2P systems are based on simple exact keyword matching lookup and cannot meet the performance requirements of emerging applications. These applications often require complex range queries or content based similarity search on data such as images, text and video. The content of nodes is described by features, particularly physical image features that are represented in multidimensional data space.

Many search techniques have been proposed for P2P systems relying on their underlying overlay infrastructure. Flooding [2] is one of the first search techniques employed in P2P systems where each peer broadcasts the received query to directly connected peers. A Time-To-Live (TTL) mechanism or a random walk method can be used to reduce the number of peers that are involved in processing a query and avoid overloading the network. DHT systems [10] [9] [3] organizes data in a key space for efficient data access. Unique identifiers are assigned to the peers and the data. A data object is mapped to the peer with the closest identifier. Each peer maintains a routing table composed of its neighbors' identifiers. A lookup query routed to and processed by the peer that contains the corresponding data keys. DHT techniques are efficient for complete exact match queries but perform poorly for approximate similarity search. Thus, the main challenge for that systems is to process complex queries such as similarity, approximate and range selections. This challenge was recently addressed in [8] by adding a layer on top of the existing DHT systems to process multi-attribute range queries.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 772–781, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In our previous work [6], we have presented a Hybrid Overlay Network (HON) for efficient similarity search. HON organizes both peers and data in an n-dimensional feature space based on content description. It is based on two key ideas. First it organizes and clusters peers sharing similar contents in the n-dimensional feature space to limit flooding overhead and send queries only to relevant peers. Second, it organizes and places similar data objects in relatively dense and adjacent regions of the feature space to achieve efficient processing of complex queries such as range and neighboring queries. The feature space represents particular attributes associated with data objects (e.g., color for an image, concept or keyword for text document) and is partitioned into cells obtained by dividing the range values of each feature into a number of intervals. Two data are similar if they are mapped to the same cell. The distribution of data objects over the cells defines the similarity between peers. Two peers are similar if their contents are distributed on the same sub regions of the feature space. We have presented extensive performance evaluation of similarity search quality in HON and shown that it achieves a high success rate.

The focus of this paper is on maintenance cost and fault tolerance issues of HON. Routing and localization methodologies are implemented in HON by maintaining partial routing tables in each peer, making the system very sensitive to membership changes. When peers join or leave the system, messages are exchanged to maintain the right P2P network topology. Thus, maintenance overheads and fault tolerance capabilities are important and can affect the performance of the system. The contributions of this work are two-fold: 1) we present and evaluate the HON system showing its scalability to large network size and numbers of data objects. Moreover, its adaptability to dynamic membership with low maintenance overheads. 2) we show through extensive simulations that HON efficiently routes queries along best available paths which make it resilient to peers' failures.

The remainder of the paper is organized as follows. In the next section, we give a brief description of HON. In section 3, we present the simulation setup describing the different parameters and metrics used to evaluate the system. Section 4 presents the evaluation results. Section 5 gives an overview about some related semantic-based search techniques to our approach. And finally section 6 concludes the paper.

2 HON

HON is a Hybrid Overlay Network that groups in the same clusters peers whose data objects are similarly distributed in a feature space defined by a set of features $f_1, f_2, ..., f_n$. The basic idea is to define a partition of the feature space into cells and use the distribution of data objects over the cells as the basis for defining peer similarity, creating clusters and computing query similarities to peers and clusters. Three steps are required to organize the data and peers in the feature space and create the clusters. First, the content of peers is distributed over the cells. Figure 1 shows the partition cells of a 2-dimensional feature space

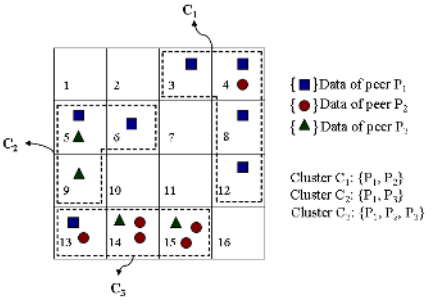


Fig. 1. Partition cells and Clusters

using the features f1 and f2 and the distribution of the contents of the peers P1, P2, P3 on the cells. Second, each peer is mapped into a set of cells containing its data objects. The mapping depends on a threshold value T. A peer is mapped to a cell only if it has a number of data objects higher than T in that cell. The last step is to create clusters by grouping the peers belonging to adjacent and dense cells in the feature space. We remind that two cells are adjacent if they share a (d-1) dimensional hyperplane, where d represents the dimension of the feature space. In addition, a cell density represents the number of objects the cell contains. The objectives of the clustering algorithm (called density-based) are to: 1) cluster peers belonging to adjacent cells for retrieving similar objects. 2) build clusters according to cells density to provide a high recall. Grouping peers within cells having high densities increases the number of retrieved similar objects. Clustering will not be discussed further in this paper. More details about the density-based algorithm and query processing can be found in our previous work [6]. The HON architecture contains two types of peers: *Super peers* and *Simple peers*. Super peers manage and maintain information about cells. Simple peers connect to super peers to process queries and have information about neighboring peers. Peers connection and disconnection to the network is processed in the following way.

Peer join: a new HON peer connects to the network and initiates the discovery of relevant super peers by broadcasting a *Join* descriptor containing its IP address and the set of cells to which its shared data belong. When a relevant super peer receives a *Join* descriptor, it updates the index tables and sends the new peer an *Accept-Join* descriptor with its IP address. Then, the new peer confirms its connection and builds the index tables with the information received from its super peers. Note that a new peer becomes the super peer of the empty cells to which it is mapped.

Peer leave: when a peer leaves the network, it sends a *Disconnect* descriptor to its super peers. A super peer receiving a *Disconnect* descriptor, removes all information related to the disconnected peer. When a super peer leaves the network, it first sends a *Disconnect* descriptor to its neighbors to initiate the

choice for a replacement. Then, one of those neighbors takeover the cells of the disconnected super peer. The peers belonging to the corresponding cells update their index tables. In the case where the super peer disconnects suddenly from the network due to failure, there is a need for keeping in each peer information about the super peers of the neighboring cells.

3 Simulation Setup

We evaluate HON using extensive simulations that focus on the maintenance cost and failure tolerance. We use then several parameters and metrics. Parameters represent a set of measurable factors, such as *Threshold*, that determines the system behavior. Metrics are measurement functions that facilitate the quantification of some particular characteristics of the system such as the *Maintenance Cost*.

3.1 Parameters

Two sets of parameters are used in our simulation: *Control parameters* and *Workload parameters*.

Control: Parameters of control are used to build the system. Their values are specified before starting a simulation, and can change for each simulation to evaluate the system in different situations. Default values of parameters are defined if no particular specifications are given for the simulation.

Parameter	Description
T	represents the threshold value used to map peers to cells. It is initialized to 0 and varies between 0 and 50% of the average data object number per peer.
N	is the number of peers in the network. It varies from 2 to 2^{16}. Its default value is 2^{16}.
G	the cell granularity which is defined by the number of partitions of features. The number of partitions takes its values between 0 and 20 and the default value is 10.
O	the number of data objects per peer. It varies from 50 to 150 and its default value is 100.
D	is the type of the data distribution. It can be Uniform or Zipfian. The default value is the Zipfian distribution which reflects the real cases.

Workload: Parameters of workload are related to the occurring events in the system. They are specified when the system is build up to simulate a mixture of peer join, departure and search operations.

Parameter	Description
β	is used to simulate peers failure. It represents the percentage of failed peers in the system. β takes its values from 0% to 70% and its default value is initialized to 20%.
QN	is the average number of queries a peer sends over the network. Its default value is 500.
QD	specify the type of queries distribution. Similarly to data distribution, query distribution can be Uniform or Zipfian. Its default value is set to a Zipfian distribution.

3.2 Metrics

In addition of improving similarity search performance at a minimum overhead [6], we analyse in this paper other aspects of HON, such as maintenance cost and fault tolerance. Thus, we define the following metrics to evaluate system scalability and fault tolerance: *Maintenance Cost*, *Load Cost* and *Failure Cost*.

• *Maintenance Cost:* is the number of maintenance messages required to build the system. When peers join the network, we compute the number of exchanged messages between peers to update indexes. We consider M the total number of maintenance messages and N the number of peers in the network. The average maintenance cost is defined by M/N.
• *Load Cost:* represents the number of hops that maintenance messages require to reach the destination. Let H be the number of hops of all maintenance messages. The average load cost is computed by H/M, where M is the number of maintenance messages.
• *Failure Cost:* we simulate the failure of a specific percentage of peers after the network is build up. The percentage of failed peers varies between 10% and 70%. We then measure the ratio of searches that fail to find existing data objects in the network. Let TQ be the total number of queries and FQ the number of failed queries. The search failure ratio is computed by FQ/TQ.

4 Evaluation Results

We focus in our simulations on evaluating the maintenance cost and failure tolerance of HON. As the system size increases, the number of peers and cells might grow exponentially. Thus high number of indexes has to be maintained and updated when peers join and leave the network. Therefore, we need to carefully plan and consider the impact of system size increase and dynamic nature of peers on maintenance costs, thus scalability and user satisfaction. In addition, peers failures are common events in P2P systems. Hence a robust system needs to be resilient to theses failures.

4.1 Scalability

Overlay maintenance cost is proportional to the number of states maintained at each peer. We study the maintenance cost after a set of peers joining events. The maintenance cost depends on several parameters that are studied in this section to analyse the system behavior. We start running a first simulation where we vary the number of peers from 2 to 2^{16}. Each peer contains between 50 and 150 data objects following a uniform distribution. The feature space is described using 5 features and divided into 1024 cells. Then, we compute the average maintenance cost of simple peers, the average maintenance cost of super peers and last the average maintenance cost of total peers constituting the system.

Figure 2 shows that when the system starts with few peers, the average maintenance cost of super peers increases while the one of simple peers is null. This can be explained by the fact that when the system starts, all the cells are empty. Therefore, the joining peers are defined as super peers to manage those cells. As a result, almost no simple peers are present at the beginning of the system life. More peers join the network, more the number of empty cells decreases, thus the probability that a new peer will become a super peer decreases. Consequently, when no more peers are designed as super peers, the super peers cost will stabilize as shown in figure 2. Meanwhile, the average maintenance cost of simple peers start increasing and stabilize till new events occur in the system.

Figure 2 shows the average maintenance cost of total peers. We notice that it goes trough three steps: *Cost Increase*, *Cost Decrease*, and *Cost Stabilization*. The average maintenance cost increases when the number of super peers is increasing, decreases when simple peers join the network with their low cost, and stabilizes when the system is completely build up.

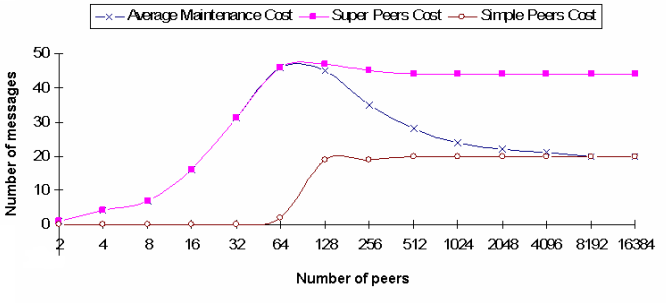


Fig. 2. The average maintenance costs

The average maintenance cost depends on several parameters. Here we study the impact of three parameters: *Data distribution*, *Cells Number* and *Threshold*. We start by the data distribution and we consider two types, the first one is a uniform distribution where each peer has equal chance to be mapped to any cell of the feature space. The second distribution follows a zipf law where peers are

mapped to few cells of the feature space. The cells have almost the same number of objects using a uniform distribution while using a Zipf distribution 80% of each peer's content is mapped to 20% of cells.

We run our simulations using a *Uniform distribution* and a *Zipf distribution* to analyze their impact on the maintenance cost. As shown in figure 3-(a), The Zipf distribution assures lower average maintenance cost than the uniform distribution. Using a Zipf distribution a peer belongs to few cells in the feature space which means that it maintains few links to other peers in the system. While, with a uniform distribution, a peer may be mapped to a large number of cells that require a higher number of maintenance messages. For example, in figure 3-(a), the average maintenance cost when the system is build up is equal to 20 messages per peer using a Zipf distribution and 30 messages per peer using a uniform distribution. In the same way, we compute the maintenance cost according to the second parameter which is the number of cells. We notice that the number of cells depends on the dimensionality and the number of feature partitions. According to the results shown in figure 3-(b), the average maintenance cost increases with the number of cells.

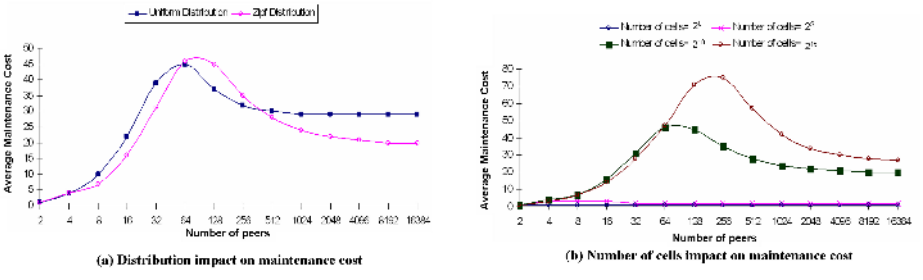
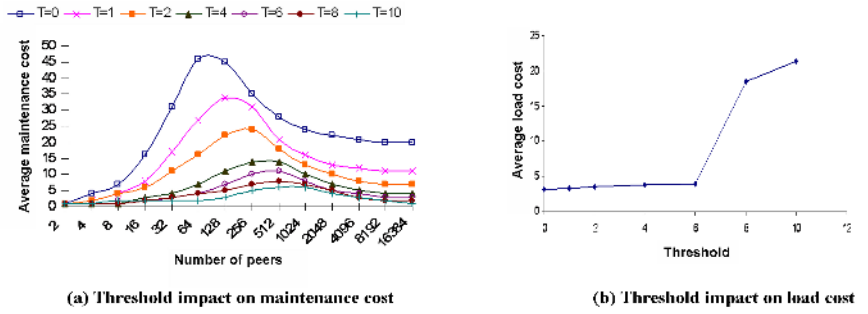


Fig. 3. Distribution and number of cells impact on maintenance cost

The third parameter we use to evaluate the maintenance cost is the threshold value. We run our experiments using a Zipf distribution. Then we vary the threshold value used to map peers to cells and observe the system behavior. Figure 4-(a) shows that the average maintenance decrease when the threshold value increases. For example, when the threshold T increases from 0 to 10, the maintenance cost decreases from 20 to 2 messages per peer. It means when the threshold increases, it reduces the number of cells to which a peer can be mapped. Therefore, peers have fewer indexes to build and to update which reduce significantly the maintenance cost. On the other hand, when the threshold value increases, the load cost increases as shown in figure 4-(b). More hops are required when the peer gets less number of connections to other peers in the network by increasing the threshold value. The results presented in 4-(b) show that a threshold T=0 provides an average load cost equals to 3 hops per message, or a threshold T=10 provides a load cost equals to 21 hops per message.



(a) Threshold impact on maintenance cost (b) Threshold impact on load cost

Fig. 4. Threshold impact on maintenance and load costs

4.2 Tolerance to Failures

Each peer maintains a *state* parameter set to 1 or 0 to indicate respectively if the peer is online or offline. A failure simulation is given by setting the *state* of β peers to 0 and then start a set of search operations to compute the search failure ratio. We remind that β represents the percentage of failed peers and takes its values between 10% and 70%. We show in the following that HON is resilient to failures in spite of its hierarchical architecture. Since there are many different paths between two points in the feature space, when one or more of peer's neighbors fail, this peer can still route along the next best available path.

The fault tolerance of HON system depends on three parameters: *Data distribution*, *Threshold* and *Cells Granurality*. We consider first the data distribution parameter using uniform and Zipf distributions. According to figure 5-(a), we note that a uniform distribution improves the fault tolerance of HON system because a peer maintains a large number of links which increases the probability to reach the required destination when peers failures occur. For example, as shown in figure 5-(a), using a uniform distribution the search failure ratio reaches only 12% with 70% of failed peers, while it reaches 49% using a Zipf distribution.

The second parameter that has a great impact on the fault tolerance in HON is the threshold value. A high threshold reduces the number of links maintained per each peer. Thus, the increase of the threshold value implies an increase of the search failure ratio. As shown in figure 5-(b) using a Zipf distribution and a threshold T=0, 50% of failed peers results in a 37% of failed queries, while a T=6 provides a search failure ratio equals to 92%.

The last parameter that we studied to measure the fault tolerance is the cells granularity. Low granularities group a high number of peers in one cell which increase the probability to find data objects with lower hops number. Therefore, the search process has a low probability to fail. In figure 5-(c), we notice that a cell granularity equals to 40 assures a null search failure ratio and a cell granularity equals to 5 provides a search failure ratio of 71% with 70% of failed peers.

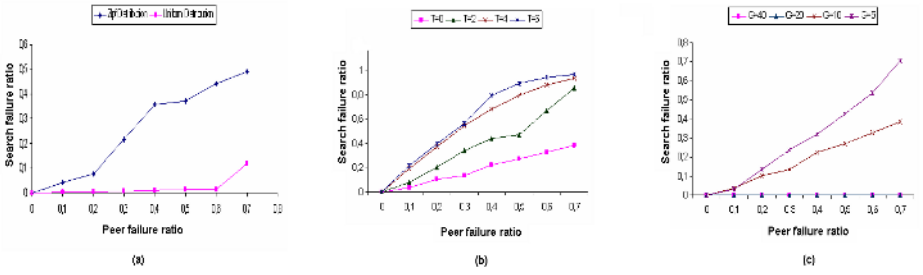


Fig. 5. Fault tolerance in HON

5 Related Work

Several deterministic semantic search approaches based on high dimensional space description of peers' content have been proposed. pSearch was the first system to allow decentralized, deterministic and non-flooding P2P information retrieval based on contents and semantics [1]. The main idea of pSearch is to store information of documents in DHT-based overlay network based on their representations. pSearch is based on CAN system[10] and uses latent Semantic Indexing LSI to generate semantic vectors for each document and query. These semantic vectors are used as index keys to store documents and route queries in the CAN space. pSearch aims to avoid the scalability problems of systems that are based on centralized indexing or index/query flooding. Even though each peer in pSearch maintains a large number of states (20), the search failure ratio grows rapidly with the number of node failures.

A similar approach to pSearch called MURK (Multi-Dimensional Rectangulation with Kd-trees) have been proposed by *Ganesan et al.* It uses a multi-dimensional data space that is partitioned into zones, where each zone is managed by one peer. A key difference between pSearch and MURK is that when a new pSearch joins a zone managed by an existing peer, that zone is divided equally between the two peers. Nevertheless, MURK splits zones into two parts of equal loads. In addition, the number of dimensions used by pSearch is governed by the dimensionality of the data while it is based on routing considerations in MURK. *Ganesan et al* focused on studying data locality properties and routing costs.

Li et al [7] have proposed a Semantic Small World (SSW) approach to facilitate efficient semantic based search in P2P networks. It is based on a semantic space where peers are clustered according to the semantics of their local data. These peer clusters are then self-organized into a small world network to assure an efficient search performance with low maintenance overhead. *Li et al* [7] have shown through extensive simulations that SSW is much more scalable to very large network sizes and very large numbers of data objects compared to pSearch. In addition, SSW assures good fault tolerance properties.

The multi-dimensional approaches presented above use mainly a maximum size M to define the boundaries of the feature space partitions. M represents

the number of peers within the partition and is set to 1 in pSearch and MURK approaches. In HON, we use predefined partitions of the feature space to perform the similarity search giving more precise description of peers' content. The search is cell-based which assure an efficient accuracy using high granularities.

6 Conclusion

Similarity search plays a key role in information sharing in P2P systems. We have presented the main characteristics of the Hybrid Overlay Network (HON), a P2P system that organizes data and peers in a multidimensional feature space to allow efficient data search. We have evaluated HON using extensive simulations that focus on the maintenance cost and failure tolerance. We have shown the scalability of HON to large network size and numbers of data objects. Moreover, its efficiency to route queries along best available paths which make it resilient to peers' failures.

Our ongoing work focuses on studying the high dimensionality problems in HON, and the load balancing issues. Moreover, we are further tuning the performance of HON to measure the efficiency of the density-based algorithm in the dynamic environment of P2P networks.

References

1. C.Tang, Z. Xu, and M. Mahalingam. Peer-to-peer information retrieval using self-organizing semantic overlay networks. In Proceedings of ACM SIGCOMM, pages 175-86, 2003.
2. Gnutella. http://www.gnutella.com, 2003.
3. I.Clarke, O.Sandberg, B.Wiley, and T.W.Hong. Freenet: A distributed anonymous information storage and retrieval system. In Lecture Notes in Computer Science, 2009:311-320, 2001.
4. I.Stoica, R.Morris, D.Karger, M. F.Kaashoek, and H.Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM, pages 149-160, 2001.
5. JXTA. http://www.jxta.org/, 2004.
6. M. Kacimi and K. Yetongnon. Density-based clustering for similarity search in a p2p network. In proceedings of the 6th IEEE Symposium on Cluster Computing and the Grid (CCGrid'06), pages 57-64, 2006.
7. M. Li, W. Lee, and A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In proceedings of the 12th IEEE International Conference on Network Protocols (ICNP'04), pages 228-238, 2004.
8. P. Rosch, K.-U. Sattler, C. Weth, and E. Buchmann. Best effort query processing in dht-based p2p systems. Proceedings of the 1st IEEE International Workshop on Networking Meets Databases (NetDB), 2005.
9. A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), pages 329-350, 2002.
10. S.Ratnasamy, P. Francis, M. Handley, R. Karp, and S.Shenker. A scalable content-addressable network. In Proceedings of ACM SIGCOMM, 2001.

Matrix-Based Programming Optimization for Improving Memory Hierarchy Performance on Imagine*

Xuejun Yang, Jing Du, Xiaobo Yan, and Yu Deng

School of Computer, National University of Defense Technology, Changsha 410073, China
{xjyang, jingdu}@yahoo.com.cn

Abstract. Despite Imagine presents an efficient memory hierarchy, the straightforward programming of scientific applications does not match the available memory hierarchy and thereby constrains the performance of stream applications. In this paper, we explore a novel matrix-based programming optimization for improving the memory hierarchy performance to sustain the operands needed for highly parallel computation. Our specific contributions include that we formulate the problem on the Data&Computation Matrix (D&C Matrix) that is proposed to abstract the relationship between streams and kernels, and present the key techniques for improving the multilevel bandwidth utilization based on this matrix. The experimental evaluation on five representative scientific applications shows that the new stream programs yielded by our optimization can effectively enhance the locality in LRF and SRF, improve the capacity utilization of LRF and SRF, make the best use of SPs and SBs, and avoid index stream overhead.

1 Introduction

Imagine is a programmable stream processor which implements an efficient memory hierarchy including several local register files (LRFs), a 128 KB stream register file (SRF) and off-chip DRAM to sustain computation on 48 arithmetic units arranged as 8 SIMD clusters [1][2][3]. Each LRF relates to a 256-word scratchpad unit (SP) used for local arrays and each SRF bank contains 8 stream buffer (SB) banks used to interface between the SRF storage and the 8 clusters [4]. Fig.1 diagrams the bandwidth hierarchy for the Imagine [5]. The stream applications on Imagine are structured as some computation kernels that operate on sequences of data records called streams [6][7]. However, most scientific applications exhibit multiple loops iterating over the same large array. It is a simple streaming method to look upon each inner loop as a separate kernel and each array as a stream. Unfortunately this straightforward coding of scientific applications does not match the available memory hierarchy on Imagine that constrains the performance of the stream applications due to the overhigh compute rate [8]. Therefore, it is necessary to explore the programming optimization for improving the utilization of the memory hierarchy.

* This work was supported by the National High Technology Development 863 Program of China under Grant No. 2004AA1Z2210.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 782 – 793, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Memory hierarchy optimization aims at achieving high utilization of the bandwidth hierarchy. The major challenge in stream programming for improving the bandwidth utilization is the utilization of the underlying hardware. First, the utilization of LRF bandwidth is limited to the kernel locality, the LRF capacity, and the usage of SPs. Similarly, the SRF bandwidth utilization is affected by the SRF locality, the SRF capacity, and the occupancy factor of SBs. Another significant aspect is the DRAM bandwidth utilization, which is decided by the application of index streams. In this paper we explore a novel matrix-based programming optimization for fully exploiting the utilizations of the above aspects to improve the memory hierarchy performance. Our specific contributions include that we formulate the problem on the Data&Computation Matrix(D&C Matrix) that is proposed to abstract the relationship between streams and kernels, and present the key techniques for improving the bandwidth utilization of LRF, SRF and DRAM based on this matrix. The experimental evaluation on ISIM simulation of Imagine [9][10] shows that the optimizing stream programs can effectively enhance the memory hierarchy performance.

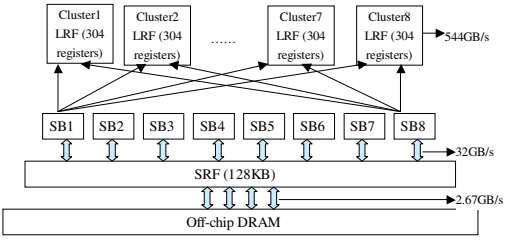


Fig. 1. The bandwidth hierarchy of Imagine

2 D&C Matrix

Our approach is based on building a matrix called the Data&Computation Matrix (D&C Matrix) for a given program shown in Fig. 2. This Matrix shows the access pattern of every array that is yielded by each iteration of all loops in the program accesses. Each raw of the D&C Matrix represents an array and each column of this matrix describes the access pattern of a loop. Suppose D_i represents a sequential layout of the array in the i^{th} row and L_j denotes the loop in the j^{th} column, the item in the i^{th} row and the j^{th} column position of the D&C Matrix corresponds to a mapping denoted as m_{ij}: $D_i \rightarrow I$ such that I is an iteration vector which presents computation order according to data order. In other words, $m_{ij}(d)$ for $d \in D_i$ is the iteration numbers in L_j that access d. Note that when each array in the nest is referenced many times, the mapping m_{ij} maps multiple data to multiple iterations. The right part of Fig. 2 gives an example of this mapping such that $m_{ij}(c)=\{x, y\}$, $m_{ij}(d)=y$ and $m_{ij}(e)=z$ for $c, d, e \in D_i$ and $\{x, y\}, \{y\}, \{z\} \in I$. To afford facilities for clarifying our approach, we express the reverse mapping of m_{ij} as $m_{ij}^{-1}(y)=\{c, d\}$. To explain our technology, it is necessary to introduce the following definitions.

Definition 1. Suppose that two iterations x and y of a loop access the same data, then the computation distance Cdistance(x,y) is defined as the number of iterations between x and y such that Cdistance(x,y)=y-x.

Definition 2. Suppose that data c and d are accessed successively, then the data distance Ddistance(c,d) is defined as the interval between the two data layouts such that Ddistance(c,d)=d-c.

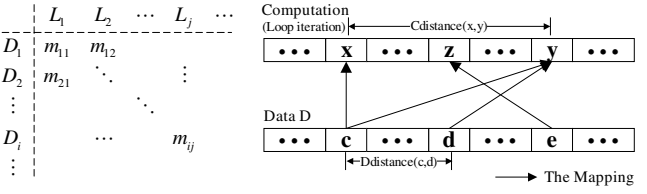


Fig. 2. The D&C Matrix and the mapping in the matrix

Each item in the D&C matrix is a mapping that presents some significant information of the access pattern of streams, including the temporal locality, the spatial locality, the access order, and the basic stream organization. For instance, data D in the right part of Fig. 2 presents the successive layout like stream layout so that we can loop upon D as a basic stream. The Cdistance(x,y) expresses the temporal locality of record c and Ddistance(c,d) denotes the spatial locality of stream D. Furthermore, we treat loop iteration spaces unrolling as the stream organization pattern, that is, the data sequence accessed by all the ordinal iterations can be organized as a stream. To clarify distinctly, we formulate this approach of stream organization as follows where ORG(i, j) is the stream organization of the i^{th} array accessed by the j^{th} loop in the D&C Matrix, the symbol "Σ^{+}" denotes the connection of different data sequences, max(x) is the maximum iteration of the loop body.

$$ORG(i,j) = \sum_{x=0}^{\max(x)}{}^{+} m_{ij}^{-1}\left(x \mid x \in I\right) \tag{1}$$

Thus, the layout of the basic stream D is important for it affects the stream organization involving the amount and the stride of index streams [11]. For example, if the basic stream D is organized as Fig.2, it need to derive a index stream as (c,e,c,d) which presents large index stride; while if the basic stream D is organized as (c,e,d), the index stream is organized as small index stride compared with Fig. 2 and thereby reducing the overhead of DRAM reordering. By analyzing the D&C Matrix, form the basic streams according to the least common array region of the most time-consuming loops. The basic stream can also be varied dynamically. We formulate the basic stream layout of each array as follows, where BAS(i) denotes the basic stream layout of the i^{th} row array in the D&C Matrix, f represents the time-consuming factor involving the invoking frequency and the running time, which shows the importance of each loop for deciding the basic stream layout.

$$BAS\ (i) = \forall j\left(\cap \left(ORG\ (i, j) \cdot \frac{1}{f} \right) \right) \tag{2}$$

3 Stream Programming for Memory Hierarchy Optimization

To improve the memory hierarchy performance of Imagine, we propose a novel matrix-based programming optimization.

3.1 Improving LRF Bandwidth Utilization

The utilization of LRF bandwidth is limited to the kernel locality that is affected by the temporal locality, the spatial locality, the LRF capacity, and the usage of SPs. Enhancing the utilization of these factors can provide high LRF bandwidth.

3.1.1 Enhancing Temporal Locality in Kernel

The temporal locality in kernel is achieved if each record is accessed many times continuously in LRF. Enhancing LRF temporal locality can increase the computational intensity [12]. Thus we propose LRF temporal locality optimizations by matrix transformations and reducing the computation distance in the D&C Matrix.

Given the j^{th} column loop with the i^{th} row stream of a kernel in the matrix, we provide the following formula for fine kernel temporal locality based on the D&C Matrix, where x is a arbitrary iteration in the j^{th} column loop, D(K) presents the streams of kernel K, and L(K) presents the loops of kernel K.

$$\forall i \forall j \left(\forall x \in L_j \left(m_{ij}^{-1}(x) = m_{ij}^{-1}(x \pm 1) \right) \mid D_i \in D(K) \cap L_j \in L(K) \right) \tag{3}$$

First, aiming at increasing more operations per memory access, we restructure all the loops based on the D&C Matrix to centralize all the computations that perform on the same stream into a large kernel. To implement this optimization, we perform matrix distribution and matrix fusion on the loops that satisfy the following formula. Then yield a new D&C Matrix with fine computational intensiveness.

$$\forall j_1 j_2 \in I \left(\exists a \in D_i \left(\left(m_{ij_1}(a) \cap m_{ij_2}(a) \right) \neq \phi \right) \right) \tag{4}$$

Second, if arbitrary successive iterations in a loop access the same record of a stream, we can say this kernel exhibits temporal locality. Thus after matrix transformation, we consider reducing the computation distance in the new D&C Matrix by computation reordering to improve the kernel temporal locality as follows.

1. Eliminating loop-carried dependence

Data dependence tells us that two references point to the same LRF location, thus the computation distance can be shortened by eliminating the loop-carried dependence [13] through array expansion, code replication etc. transformations, and making dependence just exist within inner loops, which is shown in Fig. 3.

2. Tiling the computation space

If the loop-carried dependence between loops of long stream can't be converted into the loop-independent dependence, we consider tiling the computation space given in Fig. 3 for reducing the computation distance [14]. Above all, we need partition the computation space to several parts. Then change the order of these parts to shorten the computation distance between the parts. Thus the size and the order of these computation parts play an important role in kernel temporal locality.

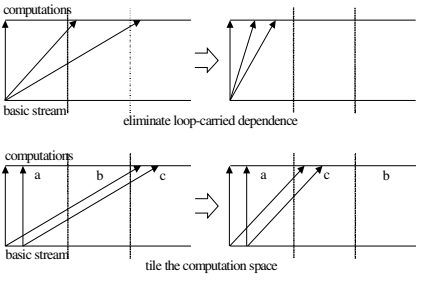


Fig. 3. Enhancing LRF temporal locality

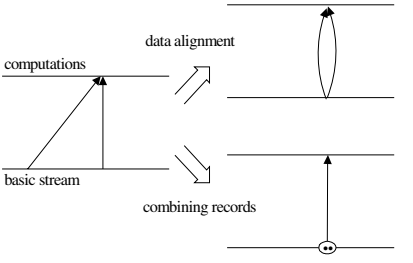


Fig. 4. Enhancing LRF spatial locality

3.1.2 Enhancing Spatial Locality in Kernel

The spatial reuse of LRF is highest when records in the LRF are accessed sequentially. In addition, having the right loop as the inner loop is critical because the inner loop determines which array dimension is accessed sequentially and the iterations of inner loop need to be placed into a cluster to enhance high spatial locality in kernel.

Given the j^{th} column loop with the i^{th} row stream of a kernel in the matrix, we provide the following formula for fine kernel spatial locality based on the D&C Matrix, where a is a arbitrary record in the i^{th} row stream.

$$\forall i \forall j \left(\forall a \in D_i \left(m_{ij}(a) = m_{ij}(a \pm 1) \right) \mid D_i \in D(K) \cap L_j \in L(K) \right) \tag{5}$$

To improve LRF spatial locality, we need to shorten the data distance in the D&C Matrix as follows through changing the records accessed by the same computation so that the neighboring records are referenced close together in time.

1. Data alignment

The approach of data alignment [15] is to align different records to the same computation by adding an extra iteration and adjusting the indices of one of the statement, and thereby the data distance can be reduced so that achieve fine spatial locality in kernel. Here is an example shown in Fig. 4. We can observe the spatial locality of the basic stream is enhanced.

2. Combining records

All items of a record are placed on a cluster to perform the same computations. So the spatial locality can be improved by combining the records referenced by the same computation as a big record according to the capacity of LRF shown in Fig. 4. At the same time, we must claim attention to save the array boundary of the big record, for the record may be as large as the capacity of LRF [16]. This idea can avoid assigning dependent data within an iteration to different clusters and make full use of LRF.

3.1.3 Improving the Capacity Utilization of LRF

To solve the locality problem of long stream, we must avoid the very latter part of a long stream reusing previous data due to the limited LRF capability, we must perform two program transformations: one side, the loop-carried dependence need be converted into loop-independent dependence for eliminating the reuse of LRF between loops of long stream; on the other side, LRF locality with limited capability can be improved by strip mining the inner loop so that a single strip length fits in LRF and then moving the loop that iterates over the strips to the outermost position.

3.1.4 Improving the Utilization of SPs

Different from the spatial reuse in cache where the cache line can make random access through index support, the spatial reuse in LRF is successive and limited to the overhead caused by SPs. The spatial locality of LRF occurs when each iteration of a loop accesses a LRF location that is adjacent to location used in the previous iteration, thus the intermediate variables produced by the previous iteration are assigned to SPs for latter iteration using. So the allocation and usage of SPs are particularly important for enhancing LRF locality. We formulize the number of SPs kept before iteration y, where NUM(X) denotes the number of sequence X.

$$\sum NUM\left(m_{ij}^{-1}(z)\right) \mid \forall i\left(\forall z > y\right)\left(\exists z\left(m_{ij}^{-1}(z) < m_{ij}^{-1}(y)\right)\right) \cap \left(D_i \in D(K)\right)\Big) \tag{6}$$

The fewest SPs are required to hold the values between source and sink of the dependence, that is minimize $\sum NUM\left(m_{ij}^{-1}(z)\right)$. Up to this point, we must reduce the dependent threshold of inner loop which denotes how many SPs would be allocated to reduce SP overhead. At the same time, to avoid the latter part of a long stream reusing the previous data due to overfull SPs, we must perform transformations involving eliminating the loop-carried dependence and tiling the computation space.

3.2 Improving SRF Bandwidth Utilization

Enhancing the utilization of the following factors can provide high SRF bandwidth: the SRF locality [17], the SRF capacity and the utilization of SBs.

3.2.1 Improving SRF Locality

The SRF locality is exposed by forwarding the streams produced by one kernel to subsequent kernels. In order to enhance SRF locality, the order of kernels in the D&C Matrix need to be reordered when the new matrix is produced. Then we bring forward optimizations for enhancing the SRF locality based on the new matrix.

1. Unifying streams between kernels

To achieve the stream reuse between successive kernels in SRF, we need to alter the streams' region to make the streams in successive kernels uniform. This idea given in Fig. 5 emphasizes on adding or reducing some additional records of certain of the streams with the variety of the corresponding computations.

2. Strip-mining streams

If some parts of a long stream can be reused between multiple kernels, we consider strip-mining the stream to enhance SRF locality. Strip-mining partitions the input

stream into segments known as strips such that all of the intermediate state for the computation on a single strip fits in the SRF. Thus multiple strips can be operated on in sequence. This method organizes streams that can be captured entirely in on-chip memory, and do not generate off-chip memory accesses.

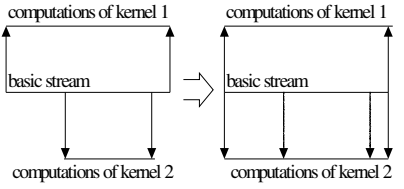


Fig. 5. Unifying streams in SRF

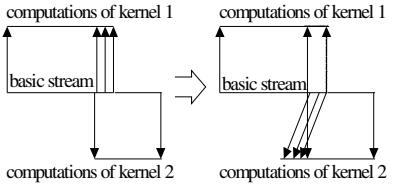


Fig. 6. Improving the SRF capacity utilization

3.2.2 Improving the SRF Capacity Utilization

Another aspect of improving the SRF bandwidth utilization is the SRF capacity utilization. We present some methods to make full use of the SRF capacity. First, we can transfer some loops in the previous kernel to the next kernel, if these loops exist data dependency with partial loops in the next kernel, which is shown in Fig. 6. This idea can reduce the producing of intermediate results to guarantee SRF capacity enough and enhance SRF reuse. The essence of this transformation is to distribute the loops in different kernels and then to fuse partial loops to a kernel based on data-centric analysis. Second, we can also use strip-mining to partition an input stream into smaller strips when this stream is larger than the SRF.

3.2.3 Improving the Utilization of SBs

With sequential SRF access, each SB is statically allocated to a single stream, that is, SRF supports eight streams transfer with each cluster at the same time. Thus, we must try our best to sustain the maximal streams to clusters. To increase the number of streams of a kernel, [5] proposes a method named stream partition according to its length or record. But this method is just suited for the programs with special streams. The more efficient method for optimizing the SBs utilization is enlarging the kernel granularity with more streams. We can use loop distribution and loop fusion [18][19] to combine loops into larger loops as many as possible to enlarge the kernel granularity by applying various transformations including privatization, alignment, replication and so on. We can also use loop scheduling introduced in section 3.2.2 to expose more streams to SBs for high performance.

3.3 Improving DRAM Bandwidth Utilization

The usage of index stream makes stream organization flexibly, but it also reduces DRAM bandwidth performance owing to too much overhead of reordering stream in DRAM and reloading the index stream to SRF. So we must avoid using index stream or reducing the length and stride of index stream for stream organization as follows.

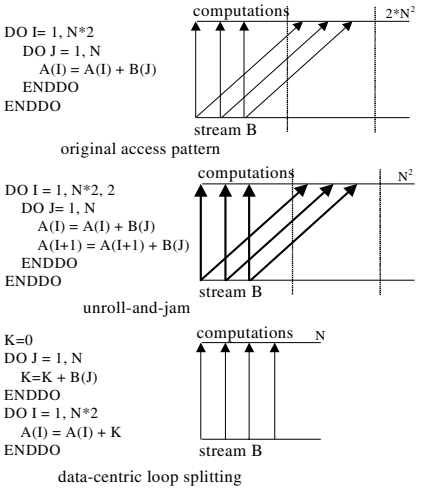


Fig. 7. Improving DRAM bandwidth utilization

3.3.1 Organizing Streams as the Basic Stream

To avoid using index stream that causes DRAM reordered overhead, we need select basic streams as operation objects of kernels. Based on the basic streams, we can choose appropriate index streams to avoid changing the loop regions and communicating between clusters accordingly. For example, if just modify a part of a big matrix, we can save this matrix as a small matrix and express this small matrix as basic stream based on access pattern, so the basic stream can be used instead of index stream to achieve good performance.

A stream is restructured as the organization of the basic stream can also reduce the index stride by some transformations like loop interchange. For example, suppose the basic stream are stored in column-major order so if the loop iterating over a row is at the innermost position, we must perform loop interchange to reduce the index stride with less overhead of index organizing.

3.3.2 Unroll-and-Jam

Performing unroll-and-jam [15] to reduce the length of index stream by improving computations per record. The essence of this transformation is to unroll the outer loop to multiple iterations and then to fuse the copies of the inner loop. As an example, consider the loop in Fig. 7. By performing this transformation, the new version of the loop performs only one load of B(J) for each two uses as follows, so the index streams of B are shortened half length from $2*N^2$ to N^2 for reducing the overhead of index stream loading.

3.3.3 Data-Centric Loop Splitting

We bring forward a new transformation to avoid index stream for higher performance compared with unroll-and-jam, which is data-centric loop splitting. We can distill the computations that reuse data with large temporal span as self-governed loop. As the previous example, the multiple loop can be split into two loops with computations on

B and A respectively due to the discontinuous temporal reuse of B(J), thus the kernel can use the basic streams of B and A without indexing overhead, which is shown in Fig. 7.

4 Experimental Results and Analysis

Five representative scientific programs are used to evaluate our matrix-based optimization named MBO on ISIM that is a cycle-accurate simulator of Imagine [9], including 171.Swim, Dfft, Transp, Vpenta and N-S.

Swim is a weather prediction program in SPEC2000, which present large data amount, irregular access pattern and few computations correspondingly. Dfft is the most time-consuming subroutines in Capao that is an application on the field of optics. It possesses small computations and fine computational intensiveness by applying butterfly algorithm. Transp that presents huge computations also comes from Capao due to its large time overhead. It exhibits loops that can be restructured to achieve high computational intensiveness, and the dependences of the loops provide the probability of enhancing locality. Vpenta is one of the kernels in NASA, which involves eight loop nests, and uses seven 2-dimensional arrays and two 3-dimensional arrays with regular data access pattern. N-S is an application of solving partial differential equation, and it is used widely in the field of fluid dynamics. N-S presents regular access pattern, fine data locality and large computations with invoking a great deal of mathematical functions.

Fig.8 shows the effect on kernel size by applying our MBO optimization compared with the stream programs without using MBO, as well as the number of computations per memory access and the number of kernels. We can observe the MBO optimization improves kernel granularity of the five programs. But the granularity of Swim achieves a little varying, because this program has too many data and irregular access pattern so that the loops in Swim are difficult to be distributed or combined, and results in low computations per memory access. Transp, Vpenta, N-S and Dfft can enlarge the code amount of kernels obviously, that is the computational intensiveness is enhanced accordingly except Transp. Transp that involves two imperfectly nested loops can apply MBO to implement loop distribution and loop fusion by array

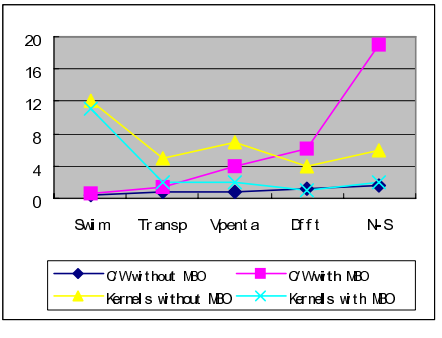


Fig. 8. The variety of kernel size

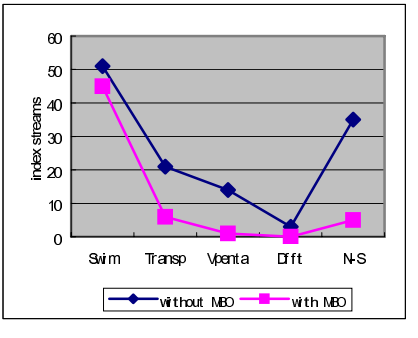


Fig. 9. The reduction of index streams

expanding effectively, however all the arrays in Transp are referenced rarely leading a little variety of computational intensiveness compared with original stream program. Different from Transp, Vpenta achieves not only fine kernel granularity but also higher computations per memory access by matrix-based optimization due to repetitive references to each array in this program. Dfft and N-S that are computational intensive applications require high computation per memory access to amortize off-chip memory bandwidth, thus the MBO optimization can centralize all computations in Dfft and N-S to a kernel so that the computational intensiveness of the two applications are increased observably.

Fig. 9 shows the reduction of index streams by applying MBO. One of the key techniques in MBO is to form appropriate basic streams so that the index streams can be reduced by appropriate program transformations based on the basic streams. But in Swim, the choice of basic stream has little effect on stream forming owing to complex data access pattern, and thus the number of index streams reduces a little. Dfft has a few data in original stream program, so the index streams are lessened a little too. The index streams of Transp, Vpenta and N-S reduce observably for achieving higher performance. In Transp, lessening the scale of original basic streams at the beginning of this program can avoid a great deal of index streams. The index stream can be eliminated in Vpenta by applying MBO when stream is short due to regular data access pattern. The speedup variety according to varying stream length of Vpenta is shown in Fig.10. We can observe the speedup is improved highly when the streams are shorter than 256, because there is no index stream by using SPs in kernel. In N-S, the basic stream reorganization plays an important role of reducing index streams.

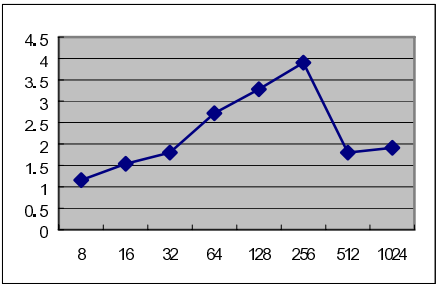


Fig. 10. The speedup for varying stream length

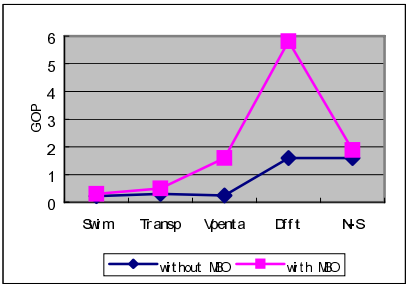


Fig. 11. Computation rate of applications

Fig.11 presents the variety of computation rate of these applications measured in the number of operations executed per second by applying MBO optimization. And illuminates the degree of memory operations and computation overlapping, with the goal of keeping all the units busy at all times. Our MBO optimization assigns all dependent data to a cluster, avoiding communication delay and memory access latency. However Swim optimized by MBO still presents overfull index steams so that memory delay can't be overlapped, resulting in low computation rate. Despite Transp and Vpenta both achieve higher LRF locality by eliminating loop-carried dependence between inner loops and shortening dependent threshold in inner loop, their computation rate are increased a little, because both low computational intensiveness of

Transp and the usage of index streams in Vpenta when streams are long make overlapping memory latency difficultly. N-S also presents high computational density by applying MBO optimization, however the computation rate of N-S is slow because it invokes inefficient mathematical kernels for many times involving sine function, cosine function, exponential function, extraction function, and exponentiation function. The high computation rate of Dfft indicates that the stream programming system delivers high computational density on this application.

Table 1 illustrates the efficiency of the program (MBO) yielded by our optimization compared with original stream program (Orig) and serial program (Seri). It is obvious that our optimization provides high speedup of Dfft, Transp, Vpenta and N-S due to fully utilize the underlying hardware. And compared with highly sensitive to memory latency of general processor, these applications can hide latency to achieve good performance. But for data intensive applications such as Swim, the speedup is low due to irregular access pattern so that our optimization can't hide memory access latency. In conclusion, Swim is not well suited for the Imagine architecture.

Table 1. Comparison of different implementation for the scientific applications

	Swim	Vpenta	Transp	Dfft	N-S
Cycles(Seri)	1.33E+11	4.11E+08	1.58E+08	1.71E+13	7.25E+09
Cycles(Orig)	8.10E+09	4.97E+07	1.98E+07	5.07E+11	4.36E+08
Cycles(MBO)	6.69E+09	1.69E+07	9.28E+06	9.71E+10	1.68E+08
Speedup (MBO vs. Seri)	2.99	3.81	2.71	26.54	3.33
Speedup (MBO vs. Orig)	1.21	2.94	2.13	5.22	2.60

5 Conclusion and Future Work

In this paper, we have presented a novel matrix-based programming optimization for improving the memory hierarchy performance to sustain the operands needed for highly parallel computation. The key techniques include that we formulate the problem on the Data&Computation Matrix (D&C Matrix) that is proposed to abstract the relationship between streams and kernels, and present the key techniques for improving the bandwidth utilization of LRF, SRF and DRAM based on this matrix. Our approach is simple and generates stream programs for scientific applications such as Swim, Dfft, Transp, Vpenta and N-S in our experiment. The experimental evaluation shows that the new stream programs yielded by our optimization can effectively enhance the locality in LRF and SRF, improve the capacity utilization of LRF and SRF, make the most use of SPs and SBs, and avoid index stream overhead.

One future work is to research more program transformations in our optimization to exploit more architectural features of Imagine so that our optimization can achieve higher performance and more wider applicability. Another is to search more scientific applications suited for stream architecture by applying our optimization.

Acknowledgements. We gratefully thank the Stanford Imagine team for the use of their compilers and simulators and their generous help We also acknowledge the reviewers for their insightful comments.

References

1. Saman Amarasinghe, William. Stream Architectures. In PACT 2003, September 27, 2003.
2. B. Khailany et al. Imagine: Media processing with streams. IEEE Micro, 21(2): 35–46,March 2001.
3. Ujval J. Kapasi, Scott Rixner, William J. Dally, Brucek Khailany, Jung Ho Ahn, Peter Mattson and John D.Owens. Programmable Stream Processors. *IEEE Computer,* pages 54-62, August , 2003.
4. Brucek Khailany. The VLSI Implementation and Evaluation of Area-and Energy-Efficient Streaming Media Processors. Ph.D. thesis, Stanford University, 2003.
5. Lifang Zeng. Fusion and Partition-Research on Memory-access-sequence Optimization. Ph.D. thesis, National University of Defense Technology, China, 2006.
6. Ola Johnsson, Magnus Stenemo, Zain ul-Abdin. Programming & Implementation of Streaming Applications. Master's thesis, Computer and Electrical Engineering Halmstad University, 2005.
7. Saman Amarasinghe et al. Stream Languages and Programming Models. In PACT 2003, September 27, 2003.
8. Nuwan S. Jayasena. Memory Hierarchy Design for Stream Computing. Ph.D. thesis, Stanford University, 2005.
9. Peter Mattson et al. Imagine Programming System Developer's Guide. http://cva.stanford.edu, 2002.
10. Abhishek Das, Peter Mattson, et al. Imagine Programming System User's Guide 2.0. June 2004.
11. Peter Raymond Mattson. A Programming System for the Imagine Media Processor. Dept. of Electrical Engineering. Ph.D. thesis, Stanford University, 2002.
12. Jinwoo Suh, Eun-Gyu Kim, Stephen P. Crago, Lakshmi Srinivasan, and Matthew C. French. A Performance Analysis of PIM, Stream Processing, and Tiled Processing on Memory-Intensive Signal Processing Kernels. In ISCA03, 2003.
13. D. Kuck, R. Kuhn, D. Padua, B. Leasure, and M. J. Wolfe. Dependence graphs and compiler optimizations. In Conference Record of the Eighth Annual ACM Symposium on the Principles of Programming Languages, Williamsburg, VA, January 1981.
14. J. Xue. Loop Tiling for Parallelism. Kluwer Academic Publishers, Boston, 2000.
15. M. J.Wolfe. High Performance Compilers for Parallel Computing. Addison-Wesley, 1996.
16. Jing Du, Xuejun Yang, et al. Scientific Computing Applications on the Imagine Stream Processor. In ACSAC06, September 6-8, 2006.
17. Jung Ho Ahn, William J. Dally, et al. Evaluating the Imagine Stream Architecture. In ISCA2004, 2004.
18. M. J. Wolfe. Optimizing Supercompilers for Supercomputers. The MIT Press, Cambridge,MA, 1989.
19. M. E.Wolf and M. Lam.A loop transformation theory and an algorithm to maximize parallelism. IEEE Transactions on Parallel and Distributed Systems, 2(4):452 –471,October 1991.

On Design and Implementation of Adaptive Data Classification Scheme for DSM Systems

Chun-Chieh Yang[1], Ssu-Hsuan Lu[1], Hsiao-Hsi Wang[1], and Kuan-Ching Li[2]

[1] Parallel and Distributed Processing Center
Dept. of Computer Science and Information Management
Providence University
Shalu, Taichung 43301 Taiwan
{g9234025, g9234024, hhwang}@cs.pu.edu.tw
[2] Dept. of Computer Science and Information Engineering
Providence University
Shalu, Taichung 43301 Taiwan
kuancli@pu.edu.tw

Abstract. Distributed Shared Memory (DSM) environment is built by using specific softwares, to combine a number of computer hardware resources into one computing environment. Such environment not only provides an easy way to execute parallel applications, but also combines resources to speedup execution of these applications. DSM systems need to maintain data consistency in memory, what usually leads to communication overhead. Therefore, there exist a number of strategies that can be used to overcome this overhead and improve overall performance. Prefetching strategies have been proven to show great performance in DSM systems, since they can reduce data access communication latencies from remote nodes. However, these strategies also transfer unnecessary prefetching pages to remote nodes. In this research paper, we focus on the analysis of data access pattern during execution of parallel applications. We propose an Adaptive Data Classification scheme to improve prefetching strategy, with the goal to improve overall performance. Adaptive Data Classification scheme classifies data according to the access behavior of pages, so that home node uses past history access patterns of remote nodes to decide whether it needs to transfer related pages to remote nodes. From experimental results, our method can improve the performance of prefetching strategies in DSM systems.

1 Introduction

Software Distributed Shared Memory (DSM) provides a convenient and effective solution for programming parallel applications. DSM systems provide the abstraction of shared address space among computers locally interconnected: private physical memory on each host is interconnected to form a global virtual memory. However, the overall performance of a DSM system is influenced by the coherence scheme. In a DSM system, data is distributed to each computing node, and thus, it needs to access

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 794 – 805, 2006.
© Springer-Verlag Berlin Heidelberg 2006

data on the remote computing nodes when executing applications. Unfortunately, some problems such as false sharing and extra communication may occur when attempting to maintain coherence.

There are a number of strategies available to improve performance of DSM systems, such as home migration [1, 6], prefetching [9], and write detection [4]. Basically, these methods assist DSM systems to achieve shorter executing time toward better performance. Prefetching permits the home host to "pre-send" data to remote hosts. Overlapping communication time and data access time can greatly reduce number of remote page faults. Unfortunately, there are some problems in prefetching strategy such as unnecessary prefetching. Consequently, we want to use data classification to prefetch precisely. Thus, performance of distributed shared memory can be improved by reducing page faults and communication time among nodes.

The main idea of our method is using the requested access sequences of home pages on each node as a class. Our method adds two components into Effective Prefetch Strategy. One component is adding DPRE as the status of pages that our method uses. This can distinguish from pages that Effective Prefetch Strategy prefetch and eliminate pages of data classification that already have been sent. Another component is established in home nodes. When page faults occur, requesting nodes decide which status of pages will be accessed, and home nodes decide if they need to transfer extra data for data classification. Then, we propose a method to improve the way of data classification and to enhance system performance. When page faults occur, our method can transfer pages that related to requested page to node that requests that page. It can reduce times of page faults when accessing data. By this way, we can reduce times of page faults and communication time between nodes.

Our DSM experimental environment is based on JIAJIA [2, 3, 5, 6, 7, 13], which is using a lazy release consistency protocol and supporting scope consistency to maintain data consistency. The global shared memory is distributed across the processors, where each processor acts as the home of a portion of the shared memory. Each shared page has an item in the global page table to record the pointer to the home host of the page, the index to the home page table, and the index to the cache page table. Our method mainly is established in Effective Prefetch Strategy. We use three applications to make our experiments: Merge, IS, and Red-Black SOR. We make our experiments to compare our Adaptive Data Classification with original JIAJIA and Effective Prefetch Strategy. From experiments, our method can improve about 9% ~ 31% of performance over original JIAJIA.

The remaining of this paper is organized as follows. In Section 2, we introduce some prefetching strategies and data classification method. In Section 3, we introduce the method that we proposed. In Section 4, the proposed strategy is evaluated by executing Merge, IS, Red-Black SOR, and LU parallel applications in a DSM system, where different performance issues among the original JIAJIA are analyzed between the proposed strategy and Effective Prefetch Strategy. Finally, brief conclusions and comments about future work are presented in Section 5.

2 Prefetching Strategies and Data Classification

In this section, we review History Prefetching, Effective Prefetch, and Adaptive Granularity.

2.1 History Prefetching Strategy

History Prefetching is utilizing temporal locality. History Prefetching Strategy predicts which pages will be used in next operation through the status of each node accesses from home node in barrier. If it deduces that the next operation for a page will be an access, then the page will be prefetched. Multiple pages can be prefetched with one message [8, 9, 13, 14]. Unfortunately, History Prefetching Strategy has some disadvantages such as Misprefetching, Accumulated Waiting Phenomenon, and Waiting Synchronization Phenomenon [9, 13, 15].

2.2 Effective Prefetch Strategy

Effective Prefetch Strategy [10, 15, 16] improves the prefetching hit rate, reduce the number of Waiting Synchronization Phenomenon and Accumulated Waiting Phenomenon. For filtering Unnecessary Prefetches, it mainly uses different memory statuses to judge if prefetching is necessary. JIAJIA manages cache pages by using the Read-Only (RO), Read-Write (RW), and Invalid (INV). Effective Prefetch Strategy adds a new status, PREF, which indicates the status of pages that Effective Prefetch Strategy prefetches in cache. When the node occur a page fault, it will check the status of the page. If the page status is PREF, it will not record that page. That means that page has already been prefetched to that node, but that node did not use that page again. So it is unnecessary to prefetch that page.

Distributing Prefetch Overhead mainly is requesting remote nodes to record the memory addresses of invalid pages before a barrier or a lock. Remote nodes will send the addresses of invalid pages to home node during barriers or locks. Therefore, home node will send prefetching data to remote requesters during barriers or locks. Home node does not manage the GETP string that records pages which remote nodes request, the INV string that records which node has invalidated pages, and other system overhead.

Load Balancing with Barrier Synchronization mainly is to improve the effect of the Accumulated Waiting Phenomenon and Waiting Synchronization Phenomenon. When nodes are asked to send prefetching data, the nodes receiving the request will execute a conditional loop with threshold. When half the nodes finish sending their prefetching data, all the senders will be forced to leave the conditional loop.

2.3 Adaptive Granularity

We refer to Adaptive Granularity, which is proposed in [11, 12], to develop our data classification method. Adaptive Granularity can transfer different message sizes

according to data types. It divides data into two kinds: bulk data and normal data [11, 12]. Normal data is using page as memory unit in DSM system, and the size of bulk data will be decided by home nodes. Then the type of data is decided by local nodes. Using original method in DSM to transfer normal data, though using the size that bulk data defined to transfer bulk data. When the false sharing occurs, it will divide data into two blocks that have equal size. It will only transfer half block of data that contains partial data that is requested by remote node. This can reduce data flow in network. It is shown as Fig. 1.

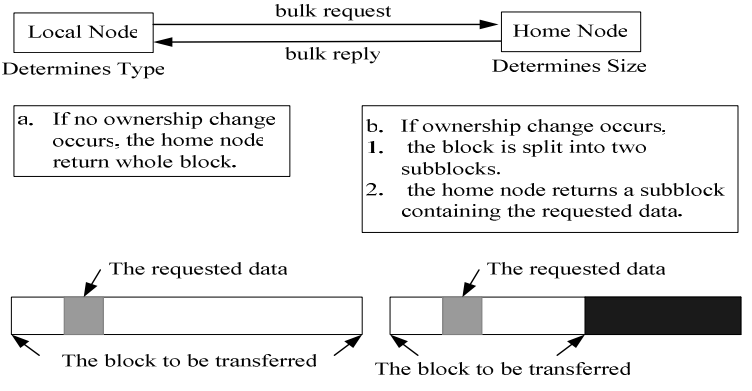


Fig. 1. Principles behind Adaptive Granularity

To support both memory replication and variable granularities, it also allocates a data structure called NCPT (Node Controller Page Table). NCPT has one record for each page of physical memory that is mapped to the local node on behalf of the home node. It contains information about how the page is mapped [11, 12].

3 Adaptive Data Classification

Our Adaptive Data Classification scheme is established in Effective Prefetch Strategy [15, 16] and is referring to the transfer message form of Adaptive Granularity [11, 12]. We hope to use data classification and prefetching strategy technique to improve performance of program executed in cluster system, reduce large amount of network traffic which is induced by maintaining data consistency in distributed shared memory, and improve the using efficiency of network.

3.1 Adaptive Data Classification

The best way to reduce remote page faults of maintaining data consistency is to let the node that wants to modify data can access data in local memory. We use this method to transfer data that maybe will be accessed to reduce remote page faults. The main idea of this method is using the requested access sequence of home pages on each node as a class. This is because the most used page of remote nodes' requests in each

home node is different. If we can find some fixed access pattern sequence, we can use this sequence to help home node to predict which pages that remote nodes will access.

Fig. 2 is the overview of our Adaptive Data Classification scheme. We use Adaptive Data Classification by the way that is similar to Adaptive Granularity, but the action of our method is different from the action of Adaptive Granularity. We observe that our method includes eight steps. In the process of dealing with page faults, we make decision if prefetching data to the node which occurring page faults. From requests of page faults, home nodes will decide which kind of pages it will need to transfer by using Adaptive Data Classification. If the page fault request is requesting page for RW, home node will not do this Adaptive Data Classification. If the page fault request is requesting page for RO, home node will do some judgments to add the page into the message and return that message to the node that occurs the page fault.

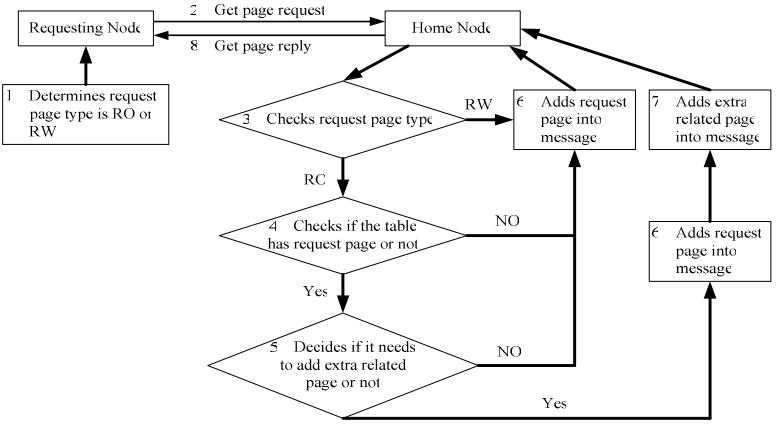


Fig. 2. Overview of Our Adaptive Data Classification Scheme

This can transfer related data at the same time that can be seen as the extending prefetching. This also can reduce times of remote accesses, times of page faults, and time of communication and computation.

3.2 Implementation

Our Adaptive Data Classification scheme adds two components with Effective Prefetch Strategy in JIAJIA. The first component is using PREF status and adding DPRE status. The PREF status means the pages that prefetched by using Effective Prefetch Strategy, and the DPRE status means the pages that carried by using our Adaptive Data Classification. This is in order to distinguish from prefetching pages of Effective Prefetch Strategy and eliminate pages of Adaptive Data Classification that already have been sent be arranged in prefetching table again. These two statuses are also used to distinguish from original statuses that JIAJIA uses. When a page fault occurs, it will check the status of that page. If the status of that page is not PREF or DPRE, it will insert this page into prefetching table. It is shown as Fig. 3.

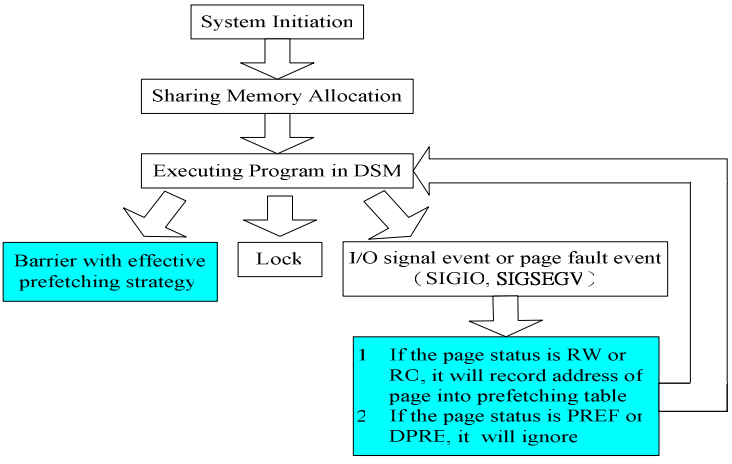


Fig. 3. Adaptive Data Classification Scheme with Effective Prefetch Strategy in JIAJIA

The second component is established in home node. It is shown as Fig. 4. Fig. 4 is the procedures of the home node deciding which pages will be transferred to which nodes by using Adaptive Data Classification scheme. When home node receives the page request, it will check if the request is for RO. If the request is for RO, home node will check if the request page is in the table, which this method established. If the request is not for RO, home node will abort this method.

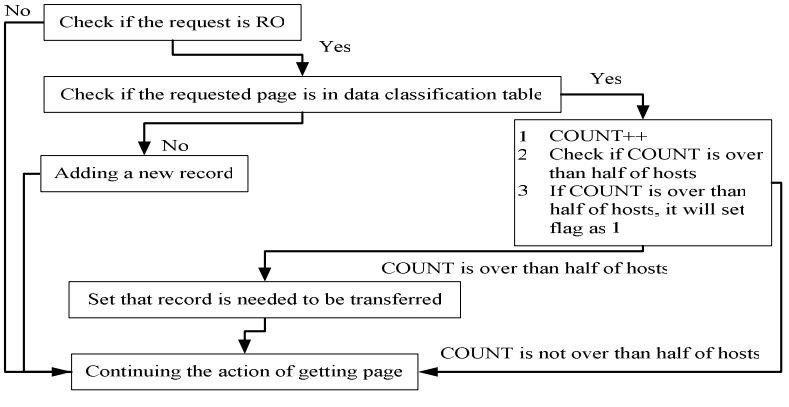


Fig. 4. Steps of Adaptive Data Classification Scheme in Home Node

The requesting node decides what action, such as RO or RW, it will do for that page, and the home node decides if it needs to transfer extra data by using Adaptive Data Classification scheme. If the page that requesting node wants to access is not in the table, home node will add the requested page into table and record times of it. If the page that requesting node wants to access is in the table, it will add times of it and

set flag as "1" which means home node needs to transfer extra data. Then home node will also check if times are over half of nodes. If the times of its record is over half of nodes, home node will set that page needs to be transferred.

When home node transfers fault pages, home node will check the flag in the table to know if there are extra pages need to be transferred. If there exists, home node will find out pages that after this fault page and set them need to be transferred and add them into message. Then the pages of Adaptive Data Classification scheme that home node transfers is not over two pages. This is because that a message packet at most can contain three pages in JIAJIA. Therefore, besides the page that the requesting node occurs page faults, one message packet can contain other two pages of Adaptive Data Classification scheme. When requesting node receives message, first it will handle fault page. If message exist extra pages, it will add extra pages into cache and set the status of that cache page as DPRE. When node accesses the status of that page is DPRE, it will use RO to handle this page. By this way, pages that Adaptive Data Classification scheme transfers will be adaptive.

4 Performance Analysis

The experimental platform consists of 8 PCs, each with one AMD Athlon 2400+ and 1GB DDR memory, all interconnected via Gigabit Ethernet, running OS RedHat 9.0 with kernel version 2.4.20. We evaluated the performance of the original JIAJIA DSM system, Effective Prefetch Strategy, and Adaptive Data Classification running four parallel applications: Merge, IS, Red-Black SOR, and LU.

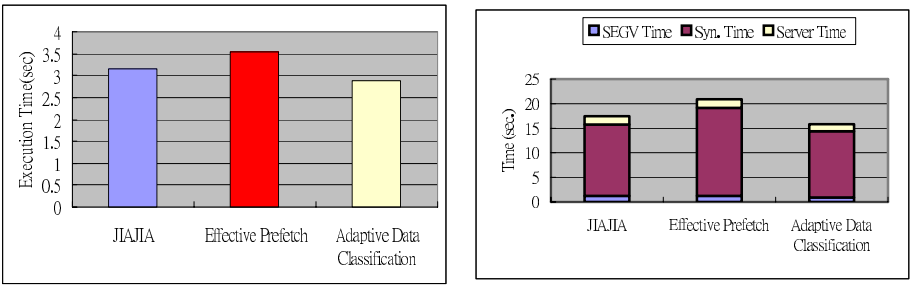
4.1 Merge

The application Merge performs the merge sort on n integers using p processors. The n integers, appeared as p sorted arrays, are held by the p processors at the starting phase. At each stage, two arrays held by adjacent processors are merged together as one sorted array by one of the processors. Hence the merging is done in $\log p$ stages. Notice that in this program, using more processors will slow down the execution, since an extra stage of merging will be introduced when the number of processors doubles [13].

Fig. 5(a) shows the execution time of Merge with JIAJIA, Effective Prefetch Strategy, and Adaptive Data Classification scheme. Our method can improve performance about 10% over JIAJIA, and improve performance about 17% over Effective Prefetch Strategy.

In Fig. 5(b), we analyze Adaptive Data Classification scheme in detail. SIGSEGV time (SEGV) represents the data miss penalty, including local and remote misses. Synchronization time (Syn.) and Server time (Server) represent the time spent on synchronization and servicing remote requests, respectively. Our scheme can reduce the SIGSEGV time, Synchronization time, and Server time of system. That means home node transfers related pages to remote nodes can help Merge be handled in DSM. This method reduces the number of remote page faults of remote nodes and the time of home

node to handle remote page faults. Because this method uses extra status to manage pages, home node can reduce unnecessary prefetching pages when processing prefetching pages. That can make home node reduce the time of processing prefetching in synchronization because it reduces amount of transferring prefetching pages.

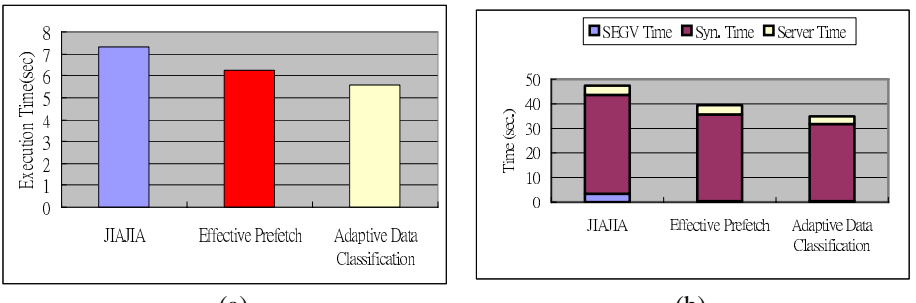


(a) (b)

Fig. 5. (a). Execution Time of Merge (b). Time Statistics of Merge

4.2 IS

IS application from NAS Benchmark Programs (NPB) ranks an unsorted sequence of keys using bucket sort. It divides up the keys among processors. There is a shared bucket for all processors and each processor has a private bucket. First, each processor counts its keys in the private array of buckets. These values in private buckets are then summed up into the shared bucket in a critical section that is protected by a lock. Finally, each processor reads the sum and ranks their keys [13].



(a) (b)

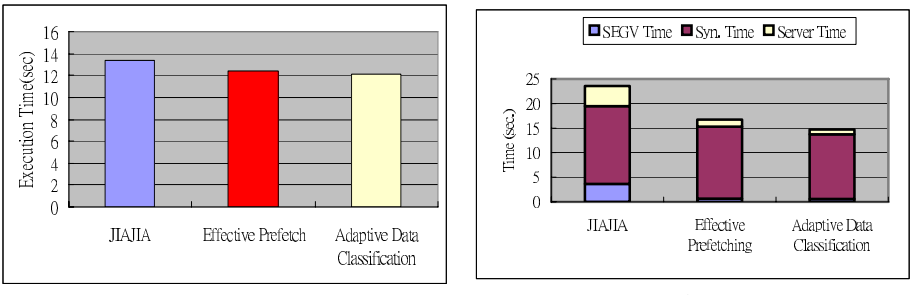
Fig. 6. (a). Execution Time of IS (b). Time Statistics of IS

Fig. 6(a) is the execution time of IS with JIAJIA, Effective Prefetch Strategy, and Adaptive Data Classification scheme. Our method can improve performance about 24% over original JIAJIA, and improve performance about 11% over Effective Prefetch Strategy.

Fig. 6(b) is the SIGSEGV time (SEGV), Synchronization time (Syn.), and Server time (Server) of IS which is using 8 computers. We can observe that prefetching can reduce time of handling page faults from Fig. 6(b). Approximately, IS is the same as Merge. Effective Prefetch Strategy and our Adaptive Data Classification scheme can reduce remote nodes occurring remote page faults, and reduce the time that home node handles remote page faults. Our method also reduces unnecessary prefetching pages of home node.

4.3 Red-Black SOR

Red-Black SOR (RBS) application is performed on two $n \times n$ matrices, one known as the red matrix, and the other called the black matrix. At each stage of the program, the values of the elements in each of the two matrices are updated according to the values of the elements in the other matrix. This routine is performed for 20 iterations. In the program, the red and black array are allocated in shared memory and divided into roughly equal size bands of rows. Each processor computes a red and a black band, and synchronizes with other processors with barriers. Communication occurs across the boundary rows on a barrier. Two barriers are used in each iteration [13].



(a) (b)

Fig. 7. (a). Execution Time of RBS (b). Time Statistics of RBS

Fig. 7(a) is the execution time of Red-Black SOR with JIAJIA, Effective Prefetch Strategy, and Adaptive Data Classification scheme. Our method can improve performance about 9% over original JIAJIA, and improve performance about 2% over Effective Prefetch Strategy. Fig. 7(b) shows the SIGSEGV time (SEGV), Synchronization time (Syn.), and Server time (Server) of Red-Black SOR that is using 8 computers. From Fig. 7(b), we can observe that both Effective Prefetch Strategy and our method can reduce these three kinds of time. In Red-Black SOR, our method only reduces SIGSEGV time and server time, and Synchronization time is almost the same as Effective Prefetch Strategy.

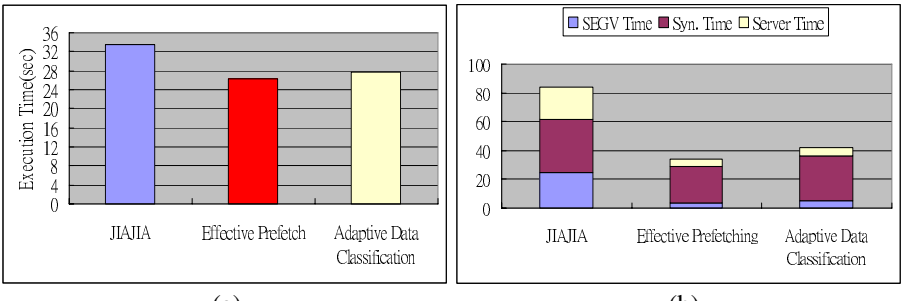
4.4 LU

LU factors a dense matrix into the product of a lower triangular and an upper triangular matrix with the block factorization algorithm. We select the contiguous

block allocation LU that allows blocks to be allocated contiguously and entirely in the local memory (home in JIAJIA) of processors that "own" them.

The algorithm factors the matrix in steps. Each step first factors the diagonal block, then the following blocks in the same column is divided by the diagonal block, and the trailing sub-matrix is updated at last. Barriers are used to separate the three phases in each factorization step [13]. To achieve good reference locality, the block size is set to 32 and the page size is set to 8192 bytes (the default page size is 4096 bytes) in the evaluation. The sizes of LU, 4096 × 4096, are run in our evaluation.

Fig. 8(a) is the execution time of LU with JIAJIA, Effective Prefetch Strategy, and Adaptive Data Classification scheme. Our method is better than original JIAJIA, but it is worse than Effective Prefetch Strategy. Our method can improve performance about 18% over original JIAJIA, but decrease performance about 8% from Effective Prefetch Strategy.



(a) (b)

Fig. 8. (a). Execution Time of LU (b). Time Statistics of LU

Fig. 8(b) shows the SIGSEGV time (SEGV), Synchronization time (Syn.), and Server time (Server) of LU that is using 8 computers. Although our method increases some overhead in Synchronization time, the performance of our Adaptive Data Classification scheme is still better than original JIAJIA. The major factor of latency is even our method can reduce prefetching pages in Effective Prefetch Strategy when prefetching, but it can only reduce a little. Our method lies on the request situation in the past. If the situation of regular page access is not precise, our method will be inefficient and may result in some latency.

4.5 Summary

From above experiments, we can observe that our Adaptive Data Classification can improve the performance of Effect Prefetch Strategy. Fig. 9 is the performance achievements when using our Adaptive Data Classification. It shows performance improvement when compared to JIAJIA. Our method shows the best overall performance. Our method can increase the accuracy of data access in Effective Prefetch Strategy, so it can reduce page faults and misprefetch. Adaptive Data Classification can improve performance about 9%~31% over original JIAJIA.

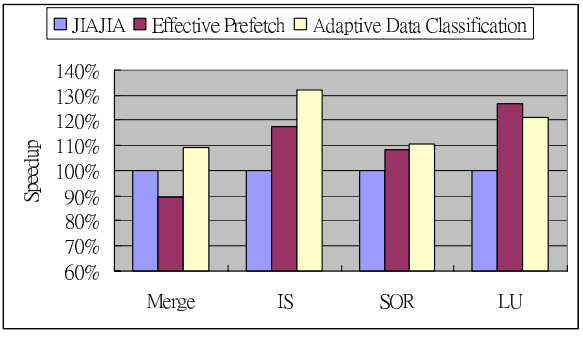


Fig. 9. Comparison of Adaptive Data Classification

5 Conclusions and Future Work

Although prefetching strategy can provide performance improvement in Distributed Shared Memory systems, there still exist some issues that need to deal with, such as misprefetch, Accumulated Waiting Phenomenon, Waiting Synchronization Phenomenon, among others. Our proposed Adaptive Data Classification scheme mainly reduces remote data access time, as also reducing the amount of same data that processors use when executing parallel applications. As result, it reduces network transmission and occurring rate of false sharing when system maintains data consistency.

The proposed method still has a number of issues, since it relies on the situation of requesting past data to help remote nodes to get pages that may be accessed in the next operation. If the situation of regular page access is not precise, our Adaptive Data Classification scheme will be inefficient for inducing extra latencies. That means it is not suitable to use the requested access sequence of home page on each node as a class in some applications. We can still improve the performance of Effective Prefetch Strategy in JIAJIA, there exist some disadvantages. We will continually improve disadvantages of this method and propose more suitable data classification scheme for every application that will greatly alleviate serious Waiting Synchronization Phenomenon in Effective Prefetch Strategy.

Acknowledgements

This paper is based upon work supported in part by National Science Council (NSC), Taiwan, under grants NSC95-2745-E-126-002-URD and NSC95-2221-E-126-006-MY3. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSC.

References

[1] T. Abe and S. Okamoto, "A Moving Home-based Software DSM System", in the *Proceedings of Communication, Computers and Signal Processing,* Vol. 1, pp. 17-20, 2003.

[2] M.R. Eskicioglu, T.A. Marsland, W. Hu, and W. Shi, "Evaluation of the JIAJIA Software DSM System on High Performance Computer Architectures", in the *Proceedings of HICSS-32 The 32nd Annual Hawaii International Conference on System Sciences,* Volume Track8, 1999.

[3] W. Hu, W. Shi, and Z. Tang. "JIAJIA: An SVM System Based on a New Cache Coherence Protocol", in the *Proceedings of HPCN '99 The High Performance Computing and Networking,* pp. 463-472, 1999.

[4] W. Hu, W. Shi, and Z. Tang, "Write Detection in Home-based Software DSMs", in the *Proceedings of the EuroPar'99*, August 31-September 2, 1999.

[5] W. Hu, W. Shi, and Z. Tang, "Reducing System Overheads in Home-based Software DSMs", in the *Proceedings of 13th International and 10th Symposium on Parallel and Distributed Processing,* pp. 167-173, 1999.

[6] W. Hu, W. Shi, and Z. Tang, "Home Migration in Home Based Software DSMs", in the *Proceedings of ACM 1st Workshop on Software DSM System,* June 1999.

[7] W. Hu, W. Shi, and Z. Tang, "Optimizing Home-Based Software DSM Protocols", *Journal of Networks, Software Tools and applications, Baltzer Science Publishers,* 4(3), pp. 235-242,2001.

[8] W. Hu, F. Zhang and H. Liu, "Dynamic Data Prefetching in Home-based Software DSM", *Journal of Computer Science and Technology,* May 2001.

[9] H. Liu and W. Hu, "A Comparison of Two Strategies of Dynamic Data Prefetching in Software DSM", *Parallel and Distributed Processing Symposium, IEEE Proceedings 15th International,* 2001.

[10] S.H. Lu, C.C. Yang, H.H. Wang, and K.C. Li, "On Design of Agent Home Scheme for Prefetching Strategy in DSM Systems", in the *Proceedings of AINA'2005 The 19th IEEE International Conference on Advanced Information Networking and Applications*, Vol. 1, pp. 693-698, 2005.

[11] D. Park and R.H. Saavedra, "Adaptive Granularity: Transparent Integration of Fine- and Coarse-Grain Communication", in the *Proceedings of Parallel Architectures and Compilation Techniques*, pp. 260-268, 1996.

[12] Y. Roh, B.H. Seong, and D. Park, "Hiding Latency through Bulk Transfer and Prefetching in Distributed Shared Memory Multiprocessors", in the *Proceedings of High Performance Computing in the Asia-Pacific Region,* Volume: 1, pp. 164–166, 14-17 May, 2000.

[13] W. Shi, "Improving the Performance of Software DSM Systems", Chinese Academy of Sciences, Institute of Computing Technology, Dept. of Computer Sciences, Doctor Thesis, Beijing, China, 1999.

[14] J.F. Tu; Y.H. Wang; L.H. Wang, "A Dynamic Data Prefetching Method of Improving the Memory Latency**",** *International Conference on High Performance Computing in the Asia-Pacific Region*, vol.1, pp. 13 -18, 2000.

[15] K.J. Wang, "On the Design and Implementation of an Effective Prefetch Strategy on DSM Systems", Providence University, Dept. of Computer Science and Information Management, Master Thesis, Taiwan, 2004.

[16] K.J. Wang, H.H. Wang, and K.C. Li, "On Design of a Prefetching Strategy for DSM System", in *PDPTA'2004 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, USA*, 2004.

TraceDo: An On-Chip Trace System for Real-Time Debug and Optimization in Multiprocessor SoC

Xiao Hu, Pengyong Ma, Shuming Chen, Yang Guo, and Xing Fang

School of Computer, National University of Defense Technology,
Changsha, Hunan, P.R. of China, 410073
xiaohu@nudt.edu.cn

Abstract. Traditional debug techniques using breakpoints and single stepping are hard to meet the requirements of debug and optimization problems related with temporal behavioral of the real-time programs in multiprocessors. In this paper an on-chip trace system TraceDo (Trace for Debug and Optimization) of a multiprocessor SoC (YHFT-QDSP) is introduced to overcome the debug challenge. Several novel methods including LS encoder, branch configuration bits and configuration instructions, have been presented in TraceDo to trace the program paths, data access and events with timestamps from four Digital Signal Processor (DSP) cores of YHFT-QDSP efficiently. The results of benchmarks show that TraceDo with LS encoder can improve the compression ratio of trace information by 27% than the best reference result on average. When using branch configuration bits, this value goes to 64%.

1 Introduction

The increasing demand of embedded systems makes embedded software more complex [10]. Developers are increasingly overwhelmed by development challenges such that system reliability is decreasing.

However, more complex software is not the only challenge to developers. The traditional debug tools are not efficient enough to keep up with new characteristics of embedded systems under time-to-market pressures.

Breakpoints and single stepping are fundamental debug methods. But they change software behavior in the real-time systems [19], and the concurrency in multi-processor systems is hard to be watched using traditional debug techniques [18]. In control systems, mechanical parts are easily damaged or out of control by stopping systems suddenly using breakpoints.

Logic analyzers do not change software behavior. But on higher integration of SoCs and processors with on-chip cache, board-level interfaces which could previously be monitored and debugged with logic analyzers are now buried in silicon [9] [14].

Software instruments and profiling are popular technique for software debug and optimizations. Information of program execution is collected by adding instructions to source code [6]. This method is intrusive and consumes excessive system resources

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 806 – 817, 2006.
© Springer-Verlag Berlin Heidelberg 2006

such as CPU cycles and memories [16]. This limits its utilizations in resource-sensitive embedded systems. Software behavior in real-time or parallel systems may be also changed by the instructions added.

When these problems have to be solved by additional area and power, on-chip trace technique is becoming popular in recent years. The leading processor-core vendors such as ARC, ARM and MIPS Technologies provide their on-chip solutions [1] [2] [7]. The IEEE-ISTO NEXUS 5001 STD also includes protocols to support on-chip trace [15].

This paper introduces TraceDo (Trace for Debug and Optimization), the on-chip trace system of YHFT-QDSP. YHFT-QDSP is a heterogeneous multiprocessor SoC integrated a commercial RISC core and four DSP cores of YHFT-DSP/700. The YHFT-DSP/700 is a high performance VLIW (Very Long Instruction Word) DSP [5]. For program debug and optimizing, TraceDo and a traditional multi-core debug system are designed for YHFT-QDSP. The traditional debug system with an enhanced JTAG port is used for breakpoints, single stepping and configuring TraceDo, etc.

TraceDo is designed for DSP cores in YHFT-QDSP. It records the run-time information of programs non-intrusively with special hardware support, and sends this information out of processors for storage and analysis. TraceDo encodes these messages into trace messages with configurable on-chip timestamps. With novel methods including LS encoder and branch configuration bits, TraceDo gets better compression ratios than reference solutions on program path trace. Besides normal trace of program path and data access, TraceDo also records pipeline stalls as event trace for optimizations. With a separated encoding scheme for stall messages, users can achieve a good trade-off between enough precision and bandwidth consumed. A NOP_config instruction is designed to configure TraceDo non-intrusively. A detailed analysis of TraceDo and other trace solutions is also present in this paper. Benchmark programs with various characters are tested in experiments.

The remainder of this paper is organized as follows. Section 2 describes TraceDo's functionality, architectural design, and advantages. This is followed in Section 3 by discussing related works. Section 4 presents experimental results of benchmark programs. Section 5 concludes the paper.

2 TraceDo

2.1 Overview

TraceDo is an on-chip trace system for debug and optimizations, as shown in Fig.1. It comprises the on-chip trace hardware, the Emulator and the software tool called Trace Analyzer embedded in the YHFT-DSP Integrated Development Environment (IDE). Trace Module of each DSP core collects and compresses trace messages. The compressed trace messages are sent to Trace Port through Trace Bus. The Emulator receives trace messages on Trace Port and attaches out-chip timestamps to them.

Trace messages are stored in DRAM buffers of Emulator or in Debug Host PC. The Trace Analyzer decompresses and analyzes trace messages to recover the behavior of programs. To reduce the bandwidth to Debug Host PC, a 24-bit timestamp counter is used in Emulator. Long time records are implemented by adding a block ID to every trace message block in Debug Host PC.

The message format of TraceDo is various lengths aligned on bytes boundary. Details of messages are present in the next section. Every type of message has a unique header in its first byte. The message length is indicated by a follow bit (F bit) or the length field in the message. The F bit indicates if there is another byte followed in this message. Such message format has several advantages: (i) various lengths aligned on bytes boundary simplify the hardware structure of FIFO and data path, in contrast to various lengths in bits. (ii)The messages in higher probability have fewer overhead bits for headers. (iii)In each type of message, with F bits, information of message length consumes fewer bits for short messages in higher probability. (iv)The massage format is independent with Trace Port.

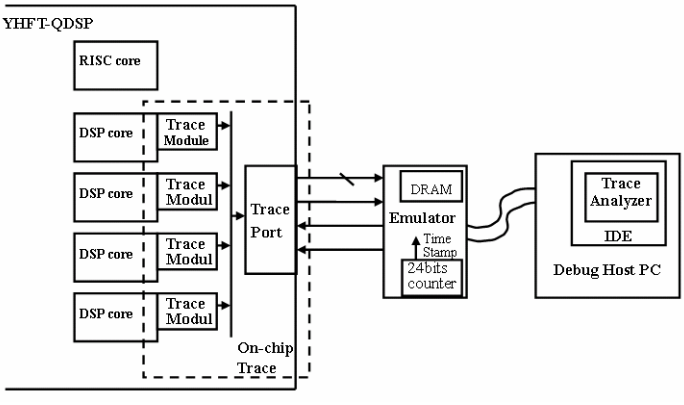


Fig. 1. Structure of TraceDo

2.2 On-Chip Trace Hardware

The structure of on-chip trace hardware is modular and scalable, as shown in Fig.2. Trace Module consists of three Trace Units collecting program path, data access and events information respectively with appropriate compression schemes. The Unit Arbitrator assembles the compressed trace messages into Trace FIFO. The Trace Bus Arbitrator reads messages from Trace FIFO and sends them into Trace Package. Trace Bus Arbitrator also adds core IDs to messages. Messages are transferred out of the chip through Trace Port. An interface wrapper is used to make signals in DSP core to compliant with interfaces of Trace Units. This wrapper simplifies the implementation of Trace Module in other specific processors.

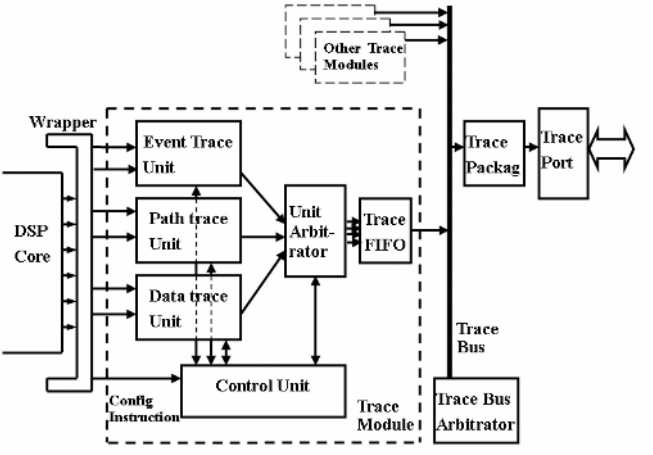


Fig. 2. Structure of On-chip Trace

2.2.1 Path Trace

TraceDo implements a Program Flow Change Model in which program path trace is synchronized at each program flow discontinuity. The program flow discontinuities are caused by branches and interrupts. The messages generated for this model are referred to as Path Trace Messages.

The Trace Analyzer can interpolate what transpires between program flow discontinuities by correlating information from Path Trace Messages with static source or object code files.

There are five types of program flow discontinuities in DSP cores: direct branches to constant addresses (BC), conditional direct branches to constant addresses (IBC); indirect branches to target address in registers (BR); conditional indirect branches to target address in registers (IBR) and interrupts to interrupt service routine. The BC causes determinate program flow change therefore it is not required by Trace Analyzer. Only one bit that indicates if an IBC is executed (taken or not taken) is required, and the constant target address can be attained from the instruction code. Target address of BR and IBR executed, taken address and target address of interrupt are also traced by default.

In this model, program code running in processors should be the same as the code analyzed by the Program Flow Change Model. Therefore, self-modifying code cannot be traced because the code in processors is not static.

A branch changes program flow just at instruction address of the branch or the following address caused by branch delay; therefore Trace Analyzer can distinguish branch taken address by only recording their temporal orders. Interrupts could change program flow at any address, so Interrupt Trace Messages with taken address information are separated from Branch Trace Messages.

Three type configuration bits are designed to control branch trace in TraceDo. One type configuration bits are four output Enable Bits in a configuration register of the Path Unit. They are designed for setting output enable mode of branch messages. The default output mode of BC is disabled, and modes of BR, IBC, IBR are enabled.

Another type configuration bit is the output Force Bit. It's set in the branch instruction code of BR/IBR. A Force Bit forces this branch to be traced out even if this type branch is disabled by its Enable Bit. Branch instruction address instead of branch target address is encoded into the message when Force Bit is valid. There is also a Degrade Bit in the instruction code of BR/IBR. The BR with its Degrade Bit set is treated as a BC by Path Units and will not be traced out by default. Similarly, the IBR with this bit set is treated as an IBC and only its execution will be traced out. The Degrade Bit is designed because many BR/IBR instructions are not inserted by the compiler for the uncertain target address, but because the target address could be out of the range of BC/IBC, or in respect that the return address of function call could be stacked. Force Bit and Degrade Bit are two reserved bits of original BR/IBR instruction code in YHFT-DSP/700. TraceDo takes advantage of these bits to communicate with on-chip trace hardware. Because the instruction code of BR/IBR has not the full target address but only the register ID, there are always some reserved bits in these instructions of 32-bit processors. In general, reserved bits of other instructions can also be used as Force Bits for communication. By reconfiguring Enable Bits, Force Bits and Degrade Bits, users can control path trace flexibly.

Five messages are designed for the Path Trace. Three of them are shown in Fig.3 as examples. Long Chart Messages and Short Chart Messages are generated by a hardware circuit named LS Encoder to record an execution history of IBC. The default mode of LS Encoder is History Mapping Mode. The history buffer of the LS Encoder is implemented as a left-shifting shift register of 6 bits. The register is always pre-loaded with a value of "000001". The "1" in this value acts as a stop bit so that the Trace Analyzer can determine which bit is the last bit of valid history. A value of "1" is shifted into the history buffer when an IBC is taken, and a value of "0" is shifted when an IBC is not taken. A not taken IBR is treated as a not taken IBC. If the five valid bits in the history buffer are all "1"s or "0"s, the LS Encoder turns from History Mapping Mode into Length Encoding Mode. The history buffer's lower 4 bits are replaced by the counter of "1"s or "0"s that will be received sequentially. A Long Chart Message is capable of recording 20 branches including the five received in History Mapping Mode. The 5th bit of the history buffer is the value it counts ("0" or "1"). The LS encoder sends out a Long Chart Message when it is ended in History Mapping Mode and sends out a Short Chart Message when it is ended in Length Encoding Mode. When the history buffer overflows or the next trace message encodes address of BR/IBR or interrupts, the LS encoder has to be ended. If the next bit received is not same to the previous one in Length Encoding Mode, the LS encoder will also be ended.

Indirect Branch Messages contain the branch target address's unique portion for a taken indirect branch. For example, if target address is 0x1010 and reference address is 0x1020, only the lower 6 bits of 0x1010 (the unique portion to 0x1020) are encoded into this Indirect Branch Message. Such compression scheme is named "XOR" in this paper. The reference address is the last address sent out or the address in the last synchronization message.

A Synchronization Message is generated every 255 branches of all types or when Trace FIFO overflows. The synchronization message contains the instruction address of a not taken branch or the target address of a taken branch. The address is compressed by removing redundant zeros in higher address bits.

Interrupt messages contain three data packets: interrupt ID (IntID), interrupt request register (MaskIntReg) and the number of instructions executed after the last branch sent out (Instr_Count). The later two packets can be disabled. With the packet of Instr_Count, the Trace Analyzer knows which interrupt is taken and where the interrupt is taken.

Short Chart Message

Header	Condition Branch Map
2bit	6bit
01	low 5 bits valid xxxxxx

Long Chart Message

Header	Branch Taken	Length
3bit	1bit	4bit
001	1 / 0	Max Counter 2^4-1+5 = 20

Indirect Branch Message

1st Byte			2nd Byte (option)		3rd Byte (option)		4th Byte (option)		5th Byte (option)	
Header 1	F 1bit	[7:2]	F 1bit	[14:8]	F 1bit	[21:15]	F 1bit	[28:22]	00000	[31:29]

F=1: no following Byte

Fig. 3. Path Trace Messages

2.2.2 Data Trace and Event Trace

Data Trace Unit records data transferred by load/store instructions. TraceDo reduces the massive data access by triggers and filters. To utilize the locality of data, the XOR compression scheme is used. There is also a Synchronization Message for Data Trace.

Events traced by TraceDo include the pipeline stalls caused by program fetch (PStall) or data fetch (DStall). With the two signals users can analyze where the CPU cycles are wasted easily. Other events such as cache missing or DMA busy can also be traced by Event Trace Unit. The encoding scheme of Event Trace is to record the counter of taken times or valid cycles of stalls at a configurable interval. The interval set in advance is defined by a number of CPU cycles or branches recorded by the Path Trace Unit. Configuration instructions can also define this interval. Such encoding scheme can achieve a good trade-off between enough precision and bandwidth consumed.

2.2.3 Trace Messages Combination and Timestamps

Trace messages from all Trace Units in YHFT-QDSP are transferred to one Trace Port by a two-level combination. The first level combination is to sort trace messages of three Trace Units into one Trace FIFO of each core. The second level is to sort trace messages of FIFOs into one Trace Port. The Unit Arbitrator in each core controls which messages will be written into the FIFO according to the predefined priorities and employments of the buffer registers in each Trace Unit. The buffer register can buffer the last two messages when the Trace Unit working for a new message.

Trace FIFO has eight input ports and one output port constructed with eight two-port register files for buffering burst messages from three Trace Units. When Trace FIFO or buffer registers overflows, a Synchronization Messages is sent out by Trace Unit.

To recover the trace messages from different cores with temporal orders, the on-chip timestamps can be configured attached to messages in Trace Units. But only the difference between this stamp and the "port time" are encoded. The "port time" is a timestamp when the message reaches to Trace Port. The difference of two timestamp is the indeterminable delay of trace message caused by FIFO. Trace Port and each Trace Module have their own 12-bit on-chip timestamp counters sharing one global clock and one reset signal. With on-chip and out-chip timestamps, user can get precise time marks of messages. The Trace Bus is 16-bit width.

2.2.4 Trace Port

Trace messages packaged are transferred to the Emulator through the Trace Port. The Trace Port has four signals, as shown by Fig.4. A data path can be configured to 4 bits, 8 bits or 16 bits in width by sharing 12 pins or 8 pins with other ports for bandwidth required by applications. An OutputActive signal indicates the active cycles on data path. An input command signal accepts simple commands from the Emulator. All these signals are synchronized with an input clock signal for working at a flexible interface frequency.

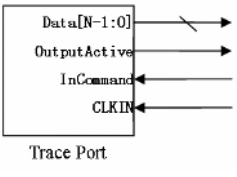


Fig. 4. Trace Port Signals

NOP_config:

1 bits	5 bits	8 bits
0: Config Command	Address	Data
1 bits	13 bits	
1: Context ID	ID	

Context Message:

1st Byte			2nd Byte		3rd Byte	
Header	F	ID	F	ID	00000	ID
0000	1 bit	[2:0]	1 bit	[9:3]		[12:10]

Fig. 5. NOP_config instruction and Context Message

2.3 Trace Configuration

The configurable functions of trace are programmed through configuration registers. All configuration registers of on-chip trace hardware can be accessed by JTAG instructions. Configuration registers of Trace Modules are also mapped into memory space of each DSP core and can be accessed by CPU instructions. Besides of load and store instructions, TraceDo supports a non-intrusive configuration instruction called NOP_config. NOP is the no-operation instruction of YHFT-DSP/700 for filling delay slots of multi-cycle instructions or empty slots in parallel instruction packages. NOP has a high instruction ratio, so they can be used by TraceDo to communicate with on-chip trace hardware. As a new instruction, NOP_config is the same as a NOP instruction in DSP core's pipeline, but Control Unit of TraceDo extracts the reserved 14 bits of this instruction as configuration commands or Context ID, as shown in Fig.5. The Context ID is sent to Trace Port as Context Messages. To synchronizing Configuration Registers and Trace Analyzer, every config write is sent to Debug Host PC.

2.4 Implementation of TraceDo

TraceDo is synthesized with timing constraints of 4ns using standard cells in 0.18um CMOS process. The area of TraceDo is 1,166,059 um^2, less than 1% of total chip area. One Trace Module requires 263,501um^2. LS Encoder only requires 2,651um^2.

3 Related Research and Analysis

An embedding debugging architecture for SoCs is introduced [9]. It integrates processor cores and development tools from different vendors into one debug environment. This architecture is not for real-time trace because it is based on scan-chains. Triggers, filters and timestamps are mentioned to support debug of multiple processors [12]. On-chip trace functions of some debugging systems are analyzed in [11]. The result shows real-time performance analysis is less well supported. An on-chip events monitor on system level is introduced in [17]. It is hard to reuse this monitor because it is based on hardware-accelerated real-time operating systems.

The CoreSight Frame of ARM core defines a multi-core debug and trace solution [2]. The CoreSight has well defined structures and functions at the cost of resources. The Embedded Trace Macrocell (ETM) in CoreSight is a real-time trace module capable of instruction and data tracing [3]. It has up to 90k gates alone [4]. The trace system of MIPS cores supports multi-pipelines. At least one trace message is sent out on every instruction executed [7].

A general framework of on-chip real time trace for multiprocessor SoCs is introduced [8]. In this framework, several interfaces have been defined to decouple the debug support and debug infrastructure from processors. The taken direct branches are indicated by setting bits in a fixed length message. The target addresses of indirect branches, data values and data addresses are all compressed by calculating difference between the current and previous values. How to deal with interrupts are not mentioned in this framework.

The IEEE-ISTO NEXUS 5001 STD defines the basic multiple core debug support for embedded processors and external tools [15]. Nexus focuses on protocol formats and doesn't define implementations. Two versions of Nexus are 5001™-1999 (Nexus1999) and 5001™-2003 (Nexus2003). Nexus defines the message format composed of a 6-bit message header and several data packets. Packets are defined in variable lengths by removing redundant zeros in higher bits of packet value. Variable length of packets and variable number of component packets make packaging trace messages into regular lengths for auxiliary port problematic. There are one or two control bits to assist external tools to partition packets and messages in data stream. The above message formats bring in redundant zeros, because two packets with variable length have to be aligned with the width of data path on auxiliary port. The first Nexus1999 implementation was in the Motorola (now Freescale) MPC565 [13]. There are still no reports about processors with Nexus2003 by now.

Every branch message of Nexus has to include a C-INT packet, because of only taken branches sent out and no special message for interrupts [13] [15]. The C-INT is

the number of instructions executed from the last taken branch. Without C-INT packets, development tools can not distinguish taken branches from other branches in source codes, or acquire taken addresses of interrupts. The redundancy comes from that addresses of all branch instructions have been known from source code but they are still indicated by I-CNT, and interrupts in relative less probability could be handled specially. Nexus2003 adds HIST packet into the indirect branch message [15]. The executions of all direct branches are mapped into the HIST. Nexus2003 adds another new message only for repeating indirect branches. But few loops constructed with indirect branches are found in benchmarks.

TraceDo records all conditional direct branches and not taken conditional indirect branches by bit mapping or length encoding. The LS Encoder switches from bit mapping to length encoding efficiently. There are separate messages for interrupts. With an "F" bit in every byte instead of a 5-bit extension header in [8], fewer bits are used to indicate the length of short messages that happen in high probability. TraceDo separates information of pipelines into program path messages and stall messages. The separation can profile pipelines with tradeoff on precision and bandwidth while ARM and MIPS have only precise mode. The quantities of trace data can be reduced significantly in TraceDo by the novel configuration of Enable Bits, Force Bits, Degrade Bits and NOP_config instructions.

In above researches, ARM and MIPS have to record every instruction executed, while MPC565, Nexus2003, [8] and TraceDo only record branches. These two methods can not be fairly compared. In the following section, we compare the compression ratios of program path trace among MPC565, Nexus2003, [8] and TraceDo.

When comparing area, the four-core solution[1] of [8] requires about 1,069,395um^2 that is less than the area of TraceDo. While including interconnects of on-chip trace hardware, areas of the two solutions are close. Because 16-bit Trace Bus of TraceDo will consume much less area than the 40-bit Point-Point bus of [8]. The message format in various lengths of TraceDo has advantages but leads to complex hardware structure on combining trace messages. LS Encoder of TraceDo only requires 2,651um^2 and hardware overheads of configuration bits can be omitted.

4 Experiments and Comparison

To evaluate TraceDo and related works, four Verilog-HDL models are constructed to describe program path trace schemes of MPC565, Nexus2003, [8] and TraceDo. The models are co-simulated with the RTL model of one DSP core in YHFT-QDSP. The max length of HIST packet of Nexus2003 is configured to 31 bits. The I-CNT packet of Nexus2003 has a max length of 8 bits, the same as that of MPC565. Synchronization messages are not considered in experiments for their low proportions and similar contributions to all results.

Ten benchmark programs of different characters are selected to test the performance of four trace schemes. Brief descriptions of benchmarks are listed in Table 1.

[1] Not include the area of Trace to Memory Unit and Break and Trigger Unit.

Table 1. Brief descriptions of benchmarks

Benchmarks	BC	IBC	IBC taken	BR	IBR	IBR taken	ALL Branch	Address covered* (Byte)	Pipeline cycles	Descriptions
uC/OSII	67092	5204	2559	46299	10	0	118605	31784	13740198	A real-time operation system
xOS	110318	121655	25631	43734	21399	12669	297106	46616	2915729	A real-time kernel
Mpeg4E	20779	750646	717182	24840	1508	21	797773	48720	16535759	MPEG-4 encoder from video products optimized
Mpeg4D	46199	1002130	926494	54104	1483	398	1103916	34932	16545326	MPEG-4 decoder from video products optimized
JpegE	1755	226033	211670	13706	28	14	241522	20772	4554919	JPEG encoder from video products optimized
MP3D	258986	1110431	56185	259729	56185	22391	1685331	56940	21205218	MP3 decoder from audio products no optimized
AdpcmD	4184	46106	37760	32	5	5	50327	6328	971304	Adpcm decoder benchmark from UTDSP
Lpc	53360	187863	53890	84725	715	76	326663	27904	4739414	LPC benchmark from UTDSP
DSP kernel	23	3844	3714	46	11	7	3924	12100**	88289	FFT, DCT, FIR, IIR and VECTORSUM from DSP fix algorithms library
float FFT	59740	105615	61343	79618	15	13	226713	12688	3216634	float FFT of 1024 points, sin, cos and bit_reverse

* The values of "Address covered" are addresses covered by the binary codes.

** This value of the "DSP kernel" benchmark is large because the code sectors are placed discontinuously for cache optimizations.

Compression efficiency of program path trace schemes is evaluated by the Compression Ratio, defined as:

$$\text{Compression Ratio} = \frac{\text{compressed size}}{\text{original size}} = \frac{\text{trace message bytes}}{\text{all taken branches} \times \text{eight bytes}}$$

Trace message bytes in this definition are the output of trace schemes. The original information before compression is instruction addresses and target addresses of all taken branches. Because of 32-bit program counter, the original size is the number of all taken branches multiplied eight bytes.

4.1 Comparison of Path Trace Schemes

Path trace messages of TraceDo are used in this comparison. The test results are shown in Fig.6. There is 27% improvement in compression ratio on average, relative to the best reference result Nexus2003. No configuration bits or NOP_config is used in this test. The improvement of TraceDo comes from LS Encoder.

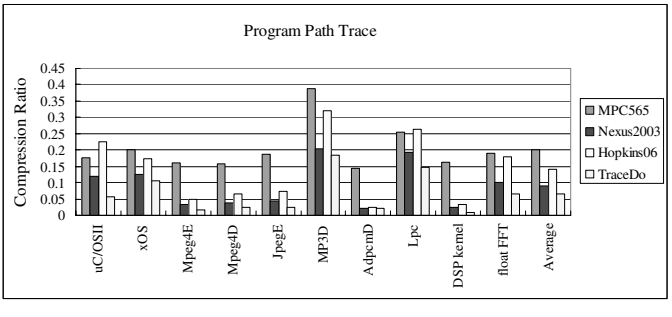


Fig. 6. Compressions of Program Path Trace. The result of [8] is labeled as Hopkins06.

4.2 Degrade Bits

To improve the compression of TraceDo, Degrade Bits are used for special indirect branches in five benchmarks. The indirect branch with the Degrade Bit set is treated as a direct branch and will not be traced by TraceDo, thus the benefit from Degrade Bits is related to the quantity of indirect branches selected. In the level-1 improvement with Degrade Bits, four indirect branch instructions executed most frequently are selected in each benchmark, and fifteen instructions are selected in level-2 improvement. The decrease of indirect branch messages is indicated in Fig.7. The Compression Ratios of Path Trace Messages are shown in Fig.8. Level-2 results in 80% improvement in float FFT and 64% on average, relative to the result of Nexus2003.

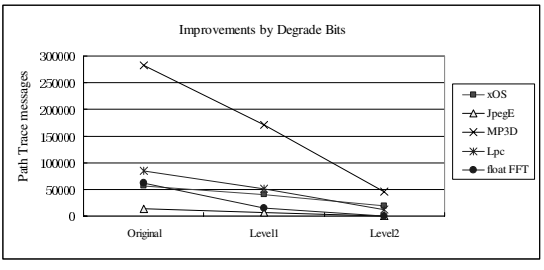


Fig. 7. Indirect branch messages decrease when selecting different Degrade Bits

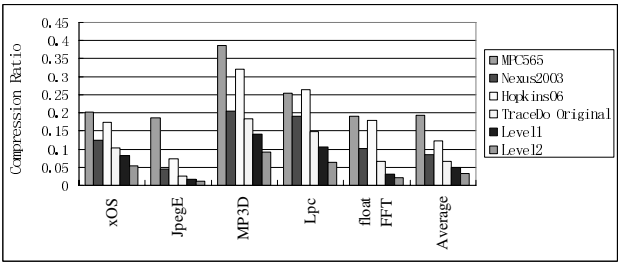


Fig. 8. Compressions of Program Path Trace with Degrade Bits. The result of [8] is labeled as Hopkins06.

5 Conclusions and Future Work

This paper introduces TraceDo, an on-chip trace system in a multiprocessor SoC. It is modular and scalable. TraceDo records information of program path, data access and pipeline stall of multicores non-intrusively. With several novel methods, TraceDo can reduce the quantity of trace messages effectively and support the tradeoff between precision and bandwidth by configuration. This paper also analyzes other trace solutions.

As future work, the Degrade Bits selected manually in benchmarks will be selected by tools automatically. As an important application of TraceDo, parallel program optimizations in YHFT-QDSP will be researched practically.

References

1. ARC International Provides New Configurable Trace and Debug Extensions to the ARC™ 600 and 700 Core Families. http://www.arc.com/news/PressRelease.html?id=227
2. CoreSight Flyer. ARM Ltd. http://www.arm.com/products/solutions/CoreSight.htm
3. Embedded Trace Macrocell Architecture Specification. ARM Ltd. http://www.arm.com/pdfs/IHI0014N_etm_v34_architecture_spec.pdf
4. Embedded Trace Macrocells Product Overvier. ARM Ltd. http://www.arm.com/miscPDFs/ 11655.pdf
5. Shuming Chen, Zhentao Li, Jianghua Wan, Dinglei Hu, Yang Guo, Dong Wang, Xiao Hu, and Shuwei Sun.: Research and Development of High Performance YHFT Digital Signal Processor, Journal of Computer Research and Development. 6(2006)
6. Rajiv Gupta, E. M., and Youtao Zhang.: Profile guided compiler optimizations. The Compiler Design Handbook: Optimizations & Machine Code Generation. CRC Press (2002)
7. The PDtrace™ Interface and Trace Control Block Specification. MIPS Technologies Inc. http://www.mips.com/content/Documentation/MIPSDocumentation/ProcessorArchitecture
8. Andrew B.T. Hopkins, K.D. McDonald-Maier.: Debug Support Strategy for Systems-on-Chips with Multiple Processor Cores, IEEE Trans. on Computers, VOL.55 2(2006)
9. R. Leatherman and N. Stollon.: An embedded debugging architecture for SoCs, IEEE Potentials, Feb/Mar (2005)
10. Edward A. Lee.: What's Ahead for Embedded Software, IEEE Computer. 9(2000)
11. Ciaran MacNamee, Donal Heffernan.: Emerging on-chip debugging techniques for real-time embedded systems, Computer & Control Engineering Journal. 12(2000)
12. A. Mayer, H. Siebert, K.D. McDonald-Maier.: Debug Support, Calibration and Emulation for Multiple Processor and Powertrain Control SoCs, DATE05 (2005)
13. MPC565 Reference Manual. Freescale Semiconductor Inc. http://www.freescale.com/files/microcontrollers /doc/ref_manual/ MPC565RM_ZIP.zip
14. Robert F. Molyneaux.: Debug and Diagnosis in the Age of System-on-a-Chip, ITC03 (2003)
15. The Nexus 5001 Forum™. http://www.nexus5001.org
16. S. Savage, M. Burrows, G. Nelson, P. Sobalvarro, and T. Anderson.: Eraser: A dynamic data race detector for multithreaded programs, ACM Trans. on Computer Systems. 11(1997)
17. Mohammed El Shobaki.: On-Chip Monitoring of Single- and Multiprocessor Hardware Real-Time Operating Systems, Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications (2002)
18. Darlene Stewart and Morven Gentleman.: Non-stop monitoring and debugging on sharedmemory multiprocessors, In Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (1997)
19. Daniel Sundmark.: Deterministic Replay Debugging of Embedded Real-Time Systems using Standard Components, PhD. Thesis, Department of Computer Science and Engineering Malardalen University Vasteras, Sweden (2004)

Automatic Performance Optimization of the Discrete Fourier Transform on Distributed Memory Computers*

Andreas Bonelli[1], Franz Franchetti[2], Juergen Lorenz[1], Markus Püschel[2], and Christoph W. Ueberhuber[1]

[1] Institute for Analysis and Scientific Computing
Vienna University of Technology
Wiedner Hauptstrasse 8-10, A-1040 Wien, Austria
{a.bonelli, juergen.lorenz, c.ueberhuber}@tuwien.ac.at
[2] Department of Electrical and Computer Engineering
Carnegie Mellon University
5000 Forbes Ave, Pittsburgh, PA, 15213, USA
{franzf, pueschel}@ece.cmu.edu

Abstract. This paper introduces a formal framework for automatically generating performance optimized implementations of the discrete Fourier transform (DFT) for distributed memory computers. The framework is implemented as part of the program generation and optimization system SPIRAL. DFT algorithms are represented as mathematical formulas in SPIRAL's internal language SPL. Using a tagging mechanism and formula rewriting, we extend SPIRAL to automatically generate parallelized formulas. Using the same mechanism, we enable the generation of rescaling DFT algorithms, which redistribute the data in intermediate steps to fewer processors to reduce communication overhead. It is a novel feature of these methods that the redistribution steps are merged with the communication steps of the algorithm to avoid additional communication overhead. Among the possible alternative algorithms, SPIRAL's search mechanism now determines the fastest for a given platform, effectively generating adapted code without human intervention. Experiments with DFT MPI programs generated by SPIRAL show performance gains of up to 30% due to rescaling. Further, our generated programs compare favorably with FFTW-MPI 2.1.5.

1 Introduction

For many important numerical problems, current compilers are not able to produce code that is competitive with hand-tuned code in efficiency. To overcome this shortcoming, a number of research efforts have developed novel methods aiming at automatic program generation, optimization, and platform adaptation [17]. Examples include ATLAS for basic linear algebra subroutines (BLAS), FFTW for the discrete Fourier transform (DFT), and SPIRAL for more general linear transforms. These and other approaches

* This work was supported by the Special Research Program SFB F011 "AURORA" and the Erwin Schrödinger Fellowship of the Austrian Science Fund FWF, and in part by DARPA through the Department of Interior grant NBCH1050009 and by NSF through awards 0234293 and 0325687.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 818–832, 2006.
© Springer-Verlag Berlin Heidelberg 2006

address the problem of automatically tuning to single processor platforms. Specifically, one goal is to tune code to a given memory hierarchy. However, with few exceptions, parallelization is still done by hand. The improvement of this situation for the DFT on distributed memory computers is the subject of this paper.

Contributions of this Paper. SPIRAL is a program generation and optimization system for linear transforms including the DFT and many others [19]. SPIRAL supports a wide range of platforms including vector architectures [7,10] and shared memory platforms [9]. In this paper we extend SPIRAL to generate MPI programs for the DFT. To do this, we identify rewriting rules that enable the automatic parallelization of FFTs given as mathematical formulas. This replaces expensive compiler analysis by simple pattern matching. In addition, we provide rules that rescale the computation to a different number of CPUs during the computation. By integrating these rules in SPIRAL's rewriting system, SPIRAL's automatic search mechanism can find the fastest among alternatives and generate DFT MPI code that is adapted to a given computing platform. We show that the generated programs benefit from rescaling for many sizes and that they compare favorably to FFTW-MPI 2.1.5. Besides performance improvement, the generation of rescaling DFT programs provides greater flexibility to the user in that it decouples initial data distribution and processor use. This flexibility is usually not provided in libraries.

Related Work. The work described in the following addresses the common problem of obtaining fast code for distributed memory platforms by automatically tuning to the platform's characteristics. The approaches range from classical compiler techniques to high level formula manipulation and program generation. The respective application domains range from general linear algebra and linear transforms to more application specific problems like quantum chemistry computations.

A compiler framework for generating MPI code for arbitrarily tiled for-loop nests by performing various loop transformations to gain *inherent* coarse-grained parallelism is presented in [14]. [18] describes the generation of collective communication MPI code by automatically searching for the best algorithm on a given system. Another empirical approach for generating efficient all-to-all communication routines for Ethernet switched clusters is used by [6].

SCALAPACK [3] is a portable library of high performance linear algebra routines for distributed memory systems following the message passing model. Built upon LAPACK, it is highly scalable on various architectures using different processor numbers. SCALAPACK requires the user to define the processor configuration and to distribute the matrix data herself.

[2] presents a parallel program generator for a class of computational problems in quantum chemistry. The input is described by tensor contractions and is manipulated using algebraic transformations to reduce the operation count. Data partitioning and memory usage optimization are performed for a specified number of processors on a given target system by using a dynamic programming search.

FFTW [11,12] is a self-adapting DFT library supporting one- and higher-dimensional real and complex input data of arbitrary size. Typically, FFTW is faster than most other publicly available FFT libraries and also compares well to vendor libraries. MPI support, i. e., MPI-FFTW, is available in FFTW 2.1.5 but not in the more recent version 3.1 [13]. FFTW requires the data to be provided in slab decomposition. It then estimates the op-

timal number of processors to use for a given computation. If this number is different from the number of CPUs the user's program runs on, FFTW requires the user to redistribute prior and after calling FFTW. If other data layouts are required, users often resort to their own custom implementations to increase performance [5,15]. Experiments [1] show that substantial portions of the runtime are spent on communication between processors. A program generation framework as presented in this paper is a step towards improving this situation in that it enables customization without programming effort.

[16] describes the extension of a sequential self-adapting package for the Walsh-Hadamard transform (WHT) to support MPI code. Different WHT matrix factorizations provided in Kronecker notation exhibit different data distributions and communication patterns. Searching the space of WHT formulas leads to the best performing factorization on a given platform. In spirit, the approach taken in [16] is similar to the framework developed in this paper.

Synopsis. Section 2 introduces the DFT and the mathematical foundation for representing its fast algorithms. Then we explain the SPIRAL system, which is the platform for our work. In Section 3, we develop the formal framework to generate MPI DFT implementation; an application of this approach to a novel method of rescaling DFT algorithms is illustrated in Section 4. We implemented the framework as extension of SPIRAL and show benchmarks of automatically generated and optimized DFT code in Section 5. The results show that rescaling provides performance gains and that our generated MPI programs compare favorably with FFTW.

2 Background: Discrete Fourier Transform and SPIRAL

Discrete Fourier Transform. The discrete Fourier transform (DFT) is the matrix vector multiplication $x \mapsto y = \mathrm{DFT}_n\, x$, where $x, y \in \mathbb{C}^n$ are the input and output, respectively, and DFT_n is the $n \times n$ matrix defined by

$$\mathrm{DFT}_n = [\omega_n^{k\ell} \mid k, \ell = 0, \ldots, n-1], \quad \omega_n = e^{2\pi\sqrt{-1}/n}.$$

The famous Cooley-Tukey fast Fourier transform (FFT) can be expressed as a factorization of DFT_n into a product of structured sparse matrices [21], namely, for $n = km$,

$$\mathrm{DFT}_{km} \rightarrow (\mathrm{DFT}_k \otimes I_m) T_m^n (I_k \otimes \mathrm{DFT}_m) L_k^n \tag{1}$$

We call (1) a *breakdown rule* since it formally represents a divide and conquer algorithm. This is emphasized by writing $\rightarrow$ instead of $=$.

In (1) we used the following notation. The $n \times n$ identity matrix is denoted with I_n; L_k^n is the stride permutation matrix defined by its underlying permutation

$$L_k^n : jm + i \mapsto ik + j, \quad 0 \le i < m,\ 0 \le j < k.$$

It is equivalent to transposing an $m \times k$ matrix stored in row-major order in memory.

Most importantly, the *tensor* or *Kronecker product* of matrices is defined by

$$A \otimes B = [a_{k,\ell} B], \quad \text{for } A = [a_{k,\ell}].$$

Finally, T_m^n is a diagonal matrix, called twiddle matrix, whose exact form can be found in [21].

Table 1. Compiling SPL into code is done by recursively using the above correspondences. x denotes the input and y the output vector. We use Matlab-like notation: `x[b:s:e]` denotes the subvector of x starting at `b`, ending at `e`, and extracted at stride `s`.

SPL construct	code
$y = (A_n B_n)x$	`t[0:1:n-1] = B(x[0:1:n-1]);` `y[0:1:n-1] = A(t[0:1:n-1]);`
$y = (I_m \otimes A_n)x$	`for (i=0;i<m;i++)` `  y[i*n:1:i*n+n-1] = A(x[i*n:1:i*n+n-1]);`
$y = (A_m \otimes I_n)x$	`for (i=0;i<m;i++)` `  y[i:n:i+m-1] = A(x[i:n:i+m-1]);`

$y = L^{km}_k x$	`for (i=0;i<k;i++)`
	`  for (j=0;j<m;j++)`
	`  y[i+k*j]=x[m*i+j];`

Recursive computation of the DFT using (1) and other FFTs (in case that n does not decompose) enables the computation of the DFT in $O(n \log(n))$ operations. Note that there is a large degree of freedom in recursing, since at each step several factorizations of n may be possible. These recursions have roughly the same operations count but different memory access patterns, which leads to different runtimes when implemented.

SPIRAL. SPIRAL [19,20] is a program generation and optimization system for linear transforms such as the DFT and many others. Its internal structure is shown in Figure 1.

The user formally specifies a transform she wants to have implemented, e.g., "DFT_{256}". First, SPIRAL recursively applies breakdown rules such as (1) to generate one out of many possible *formulas*, represented in the language SPL (signal processing language), which was informally introduced above. Namely, SPL expresses algorithms as sparse structured matrix factorizations using products, tensor products, and basic matrix such as the identity and permutations. Next, SPIRAL optimizes the structure of the formula using a formula rewriting system (see [4] for an introduction to rewriting systems). The rewriting effectively performs optimizations for the memory hierarchy [8], for vector instructions [10], or for shared memory platforms [9]. The idea is to perform these optimizations at a high level of abstraction (namely on formulas), since they are unpractical at the C code level.

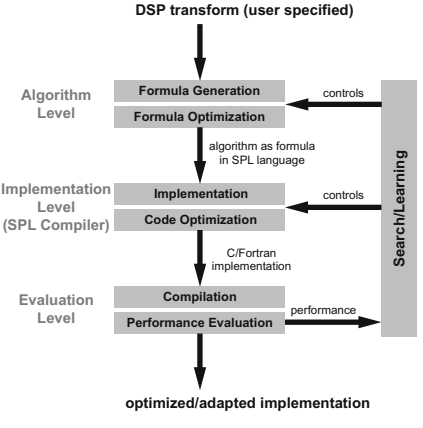


Fig. 1. SPIRAL's architecture

The obtained optimized SPL formula is then translated into C code using a special purpose compiler. This is possible since formulas have a clear interpretation as code. A few simple examples are shown in Table 1. The obtained code is further optimized and then compiled and its runtime measured.

The runtime is fed into a search engine, which drives, in a feedback loop, the formula generation process and the selection of implementation options such as the degree of unrolling. In doing so, SPIRAL effectively searches for the formula, or algorithm, that runs fastest on the given computing platform. Search strategies include dynamic programming and evolutionary search. Upon termination, the final program is output to the user.

The goal of this paper is to present first steps in extending SPIRAL to generate efficient programs for distributed memory platforms. Similar to the vector code generation and shared memory parallel code generation, we achieve this through a suitably designed extension of SPIRAL's rewriting system and the SPL compiler. This is explained in the next sections.

3 Translating Formulas into MPI Programs

In Section 2 we explained SPIRAL and its theoretical underpinning: the formula language SPL, which enables algorithm generation and optimization at a high level of abstraction. Our goal is to enable SPIRAL to generate efficient MPI implementations. To this end, we now introduce formula constructs that are translated into message passing programs by an extension of the SPL compiler, called MPI-SPL compiler. The MPI-SPL compiler is one major contribution of this paper.

Data distribution. We introduce the tag "par(p)" to express that a formula will be implemented on p processors. We assume that all distributed data vectors are block distributed, i.e., each processors' memory holds one equal sized contiguous chunk of the data vector. For instance, if a formula A_6, representing the computation $y = A_6 x$, operates on vectors of 6 data elements which are distributed across 2 processors, we write

$$\begin{pmatrix} y_0 \\ y_1 \\ \hline y_2 \\ y_3 \\ \hline y_4 \\ y_5 \end{pmatrix} = \underbrace{A_6}_{\text{par}(2)} \begin{pmatrix} x_0 \\ x_1 \\ \hline x_2 \\ x_3 \\ \hline x_4 \\ x_5 \end{pmatrix}.$$

The tag "par(2)" implies that the computation of $y = A_6 x$ is distributed across 2 processors. The elements x_0, x_1, x_2 and y_0, y_1, y_2 are stored in the memory of processor 0, while the elements x_3, x_4, x_5 and y_3, y_4, y_5 are stored in the memory of processor 1. We add a horizontal line between vector elements that reside in the local memory of different processors.

In addition, we introduce tags that express data redistribution. The tag "par($q \leftarrow p$)" expresses that the input vector x is distributed over p processors and the output vector y is distributed over q processors. This implies that the tagged formula does a redistribution from p to q processors during its computation. For instance, we denote a formula A_6 operating on vectors of 6 data elements with the input x distributed across 2 processors and the output y distributed across 3 processors by

$$\begin{pmatrix} y_0 \\ y_1 \\ \hline y_2 \\ y_3 \\ \hline y_4 \\ y_5 \end{pmatrix} = \underbrace{A_6}_{\text{par}(3 \leftarrow 2)} \begin{pmatrix} x_0 \\ x_1 \\ \hline x_2 \\ \hline x_3 \\ x_4 \\ x_5 \end{pmatrix}. \tag{2}$$

The tag "par($3 \leftarrow 2$)" implies that the computation of $y = A_6 x$ is started on 2 processors and finished on 3 processors, redistributing during computation. The elements

x_0, x_1, x_2, and y_0, y_1 are stored in the memory of processor 0, the elements x_3, x_4, x_5, and y_2, y_3 are stored in the memory of processor 1, and the elements y_4, y_5 are stored in the memory of processor 2.

Finally, we introduce the tag "par$(p \leftarrow q \leftarrow p)$," which expresses that a formula's input and output are distributed across p processors but the formula internally redistributes to q processors. For instance,

$$y = \underbrace{AB}_{\text{par}(p \leftarrow q \leftarrow p)} x \quad \text{with} \quad \underbrace{AB}_{\text{par}(p \leftarrow q \leftarrow p)} = \underbrace{A}_{\text{par}(p \leftarrow q)} \underbrace{B}_{\text{par}(q \leftarrow p)}$$

has the input x and the output y distributed over p processors, but the output of B (i.e., the input of A) is distributed across q processors.

Parallel computation. The formula construct

$$I_p \otimes A^{m \times n} = \begin{bmatrix} A^{m \times n} & & \\ & \ddots & \\ & & A^{m \times n} \end{bmatrix}, \quad A^{m \times n} \in \mathbb{C}^{m \times n}$$

is a block-diagonal matrix of p blocks of $A^{m \times n}$. The tagged formula

$$y = \underbrace{\left(I_p \otimes A^{m \times n}\right)}_{\text{par}(p)} x \tag{3}$$

expresses a p-way embarrassingly parallel computation. Each $A^{m \times n}$ operates on an independent part of x and y. The vectors $x \in \mathbb{C}^{pn}$ and $y \in \mathbb{C}^{pm}$ are distributed across p processors into p local vectors $x'_i \in \mathbb{C}^n$ and $y'_i \in \mathbb{C}^m$ with $x = x'_0 \oplus \cdots \oplus x'_{p-1}$ and $y = y'_0 \oplus \cdots \oplus y'_{p-1}$; $\oplus$ denotes the stacking of column vectors. All p processors execute the formula $A^{m \times n}$ in parallel computing $y'_i = A^{m \times n} x'_i$. Since it is the same formula in each case, (3) is easily implemented as single program multiple data (SPMD) MPI program.

Similarly, formulas consisting of diagonal matrices,

$$y = \underbrace{D}_{\text{par}(p)} x, \quad D \in \mathbb{C}^{mp \times mp} \text{ diagonal}, \tag{4}$$

can be trivially mapped to MPI programs.

All-to-all communication. Permutations express data reordering. In a distributed address space this reordering translates into explicit communication if the source and target location are in the local memory of different processors. Permutations of the form $P^{mp} \otimes I_n$, where $P^{mp} \in \mathbb{C}^{mp \times mp}$ is a permutation matrix, reorder mp chunks of n consecutive elements where m chunks reside in each processor's memory. This means that up to m messages of length n are to be sent and received per processor. Thus,

$$y = \underbrace{(P^{mp} \otimes I_n)}_{\text{par}(p)} x \tag{5}$$

encodes an all-to-all communication of p processors with message size n and the communication pattern described by P. For instance, when implementing

$$y = \underbrace{\left(L_2^4 \otimes I_2\right)}_{\text{par}(2)} x \;=\; \left[\begin{array}{cccc|cccc} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \hline \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{array}\right] \cdot \left(\begin{array}{c} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \hline x_4 \\ x_5 \\ x_6 \\ x_7 \end{array}\right),$$

processor 0 sends the message (x_2, x_3) to processor 1 and processor 1 sends the message (x_4, x_5) to processor 0.

In our example not all chunks of length m become messages. For instance, (x_0, x_1) and (x_6, x_7) stay in the memory of their respective processor. We capture this by decomposing (5) into a *local part* that copies data within the local memory of each processor and a *global part* that must be implemented using message passing. Formally, we decompose P in (5) into a sum of two matrices,

$$P = F + C,$$

and thus

$$P^{mp} \otimes I_n = (F \otimes I_n) + (C \otimes I_n).$$

Each "1" entry in P ends up either in F or C, hence the sum does not incur actual operations.

F contains all "1" entries of P within the block diagonal with blocks of size $m \times m$. It describes the addressing of all data chunks that stay within the local memory of each processor. $F \otimes I_n$ will be implemented as data copying by the respective processor.

C contains all remaining, i.e., off-blockdiagonal "1" entries. It describes the addressing of all data messages that have to be transmitted between two processors. $C \otimes I_n$ will be implemented using one send/receive pair per message.

To make the message addressing explicit, we further factor C as

$$C = SC'G \quad \text{with} \quad C',$$

where C' is a permutation matrix. This factorization is explained next. Assume, that P^{mp} requires kp messages ($k \leq m$). Then $C' \in \mathbb{C}^{kp \times kp}$ is a permutation matrix describing the message addressing. $C'_{i,j} = 1$ implies that message $(j \bmod k)$ sent by processor $\lfloor j/k \rfloor$ is message $(i \bmod k)$ received by processor $\lfloor i/k \rfloor$. G is a rectangular block-diagonal matrix of p blocks of size $k \times m$. G assigns k of the m data chunks within each processor's local memory to one of the k messages to be sent by this processor. S is a rectangular block-diagonal matrix of p blocks of size $m \times k$. It stores the k messages received by each processor at their final location within the local memory of each processor.

Analysis of S, C', and G enables highly optimized implementations like using MPI collective communication functions or implementing $y = \big(P \otimes I_n\big)x$ inplace (vector x and y share the same memory location). For instance, if C' is symmetric and S=G^T (transpose), then $C \otimes I_n$ can be implemented inplace using send-receive-replace operations. The required analysis is implemented using the techniques described in [8]. Details of the analysis are beyond the scope of this paper.

As illustrative example we parallelize L_3^9 for 3 processors. We factor L_3^9 into the local matrix F and the communication matrices S, C', and G:

$$y = \underbrace{L^9_3}_{\text{par}(3)}\, x = Fx + SC'Gx.$$

The explicit form is shown next and represents the communication addressing pattern:

$$\begin{pmatrix} y_0\\ y_1\\ \hline y_2\\ y_3\\ \hline y_4\\ y_5\\ \hline y_6\\ y_7\\ y_8 \end{pmatrix} = \begin{pmatrix} x_0\\ x_3\\ \hline x_6\\ x_1\\ \hline x_4\\ x_7\\ \hline x_2\\ x_5\\ x_8 \end{pmatrix} = \underbrace{\left[\begin{array}{ccc|ccc|ccc} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot\\ \hline \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot\\ \hline \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{array}\right]}_{\underbrace{L^9_3}_{\text{par}(3,\ \text{mpi})}} \cdot \begin{pmatrix} x_0\\ x_1\\ \hline x_2\\ x_3\\ \hline x_4\\ x_5\\ \hline x_6\\ x_7\\ x_8 \end{pmatrix} = \underbrace{\left[\begin{array}{ccc|ccc|ccc} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \hline \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{array}\right]}_{F} \cdot \begin{pmatrix} x_0\\ x_1\\ \hline x_2\\ x_3\\ \hline x_4\\ x_5\\ \hline x_6\\ x_7\\ x_8 \end{pmatrix} +$$

$$\underbrace{\left[\begin{array}{cc|cc|cc} \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ 1 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot\\ \hline \cdot & \cdot & 1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot\\ \hline \cdot & \cdot & \cdot & \cdot & 1 & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 1\\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \end{array}\right]}_{S} \underbrace{\left[\begin{array}{ccc|ccc} \cdot & \cdot & 1 & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot\\ \hline 1 & \cdot & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 1\\ \hline \cdot & 1 & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot \end{array}\right]}_{C'} \underbrace{\left[\begin{array}{ccc|ccc} \cdot & 1 & \cdot & \cdot & \cdot & \cdot\\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot\\ \hline \cdot & \cdot & \cdot & 1 & \cdot & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 1\\ \hline \cdot & \cdot & \cdot & \cdot & 1 & \cdot\\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{array}\right]}_{G} \cdot \begin{pmatrix} x_0\\ x_1\\ \hline x_2\\ x_3\\ \hline x_4\\ x_5\\ \hline x_6\\ x_7\\ x_8 \end{pmatrix}.$$

The matrix F encodes that x_0 (in processor 0's memory), x_4 (in processor 1's memory), and x_8 (in processor 2's memory) do not require communication and are moved from x to y by their respective processors. The matrix G specifies which elements of the vector x become which message. In our example the data packets are x_1, x_2 (sent by processor 0), x_3, x_5 (sent by processor 1), and x_6, x_7 (sent by processor 2). C' is a 6×6 permutation matrix encoding the send/receive addressing of the data packets. For instance, the entry $C'_{4,1} = 1$ of $C' = [C'_{i,j}]_{i,j}$ describes that message 1 sent by processor 0 (x_2) is message 0 received by processor 2. The matrix S describes the final location of the received data packets. For instance, message 0 received by processor 2 (x_2) will be stored at location y_6. Figure 2 shows the MPI corresponding implementation.

Data redistribution. Formula (2) requires different data distributions for x and y. To capture this, we generalize the idea of all-to-all communication from the previous section to data redistributions. Permutations

$$P^m \otimes I_n \quad \text{with} \quad p, q \mid m,\ P^m \text{ permutation matrix} \in \mathbb{C}^{m \times m} \tag{6}$$

reorder m chunks of data of size n. Thus,

$$y = \underbrace{\left(P^m \otimes I_n\right)}_{\text{par}(q \leftarrow p)} x \tag{7}$$

```
int proc[][] = {{1,2}, {0,2}, {0,1}},  // communication pattern
    msg[][]  = {{0,0}, {0,1}, {1,1}},
    SG[][]   = {{1,2}, {3,5}, {6,7}},
    F[]      = {0,1,2};

// parallel function, call by 3 MPI processes simultaneously
void L_9_3(double *yLocal, double *xLocal, int mpirank) {
   // output: yLocal[3], input xLocal[3]; part of x[9] and y[9]
   MPI_Request send[2], recv[2]; int i;
   yLocal[F[mpirank]] = xLocal[F[mpirank]]; // y = Fx +...
   for(i = 0; i < 2; i++){                  //     + SC'Gx
      // nonblocking send
      MPI_Isend(xLocal + SG[mpirank][i],    // source ofs
         1, MPI_DOUBLE,
         proc[mpirank][i],                  // receiving proc
         i,                                 // msg id
         MPI_COMM_WORLD, send + i);
      // nonblocking receive
      MPI_Irecv(yLocal + SG[mpirank][i],    // target ofs
         1, MPI_DOUBLE,
         proc[mpirank][i],                  // sending proc
         msg[mpirank][i],                   // msg id to get
         MPI_COMM_WORLD, recv + i);
   }
   MPI_Waitall(1, recv, MPI_STATUSSES_IGNORE);
}
```

Fig. 2. MPI program implementing $y = L_3^9 x$ on 3 processors

redistributes data from p to q processors using message size n and with the message addressing encoded in P. We apply again the approach of the last section and decompose P into local operations and communication to generated MPI code:

$$P = F + C.$$

As an example of a redistribution from 2 to 3 processors consider

$$y = \underbrace{L_2^6}_{\text{par}(3\leftarrow 2)} x = \begin{pmatrix} x_0 \\ x_2 \\ \hline x_4 \\ x_1 \\ \hline x_3 \\ x_5 \end{pmatrix} = \left[\begin{array}{ccc|ccc} 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{array}\right] \cdot \begin{pmatrix} x_0 \\ x_1 \\ \hline x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix}.$$

Processor 0 sends the message x_1 to processor 1. x_0 and x_2 stay in the memory of processor 0. Processor 1 sends the message x_3 and x_5 to processor 2 and receives x_1 from processor 0. x_4 stays in in the memory of processor 1.

Parallelization through formula rewriting. Above we introduced formula constructs that can be implemented as parallel computation or as communication. Products of these formulas can be implemented as a sequence of parallel communication and communication steps. This gives rise to the following definition.

Definition 1 (Parallelized formula). *Formulas of the form* (3)*,* (4)*,* (5)*,* (7)*, and products of these formulas are called parallelized. Parallelized formulas can be implemented using MPI.*

However, not all formulas are parallelized. For instance, the right-hand side of (1) is not a parallelized formula. Thus, we introduce a set of rewriting rules to use SPIRAL's rewriting system to transform formulas into parallelized formulas. This rule set is summarized in Table 2 and is one of the contributions of this paper. The rule set is designed for the generation of DFT MPI code. Using this rule set, SPIRAL can automatically parallelize formulas for the DFT at a high level of abstraction.

As a small example of the workings of the rewriting system consider

$$y = \underbrace{(I_m \otimes A_n)}_{\text{par}(p)} x. \tag{8}$$

(8) is not parallelized for $m \neq p$. Assuming $p|m$, the application of rule (13) transforms (8) into

$$y = \underbrace{\big(I_p \otimes (I_{m/p} \otimes A_n)\big)}_{\text{par}(p)} x$$

which matches (3) and is thus parallelized in the sense of Definition 1. A more elaborate example showing the parallelization of a DFT_{mn} and rescaling it from p to q processors is given in the next section.

Table 2. Parallelization and rescaling rewriting rules

$$\underbrace{AB}_{\text{par}(p)} \to \underbrace{A}_{\text{par}(p)}\,\underbrace{B}_{\text{par}(p)} \tag{9}$$

$$\underbrace{A}_{\text{par}(p)} \to \underbrace{A}_{\text{par}(p\leftarrow q\leftarrow p)}, \quad q|p \tag{10}$$

$$\underbrace{AB}_{\text{par}(p\leftarrow q\leftarrow p)} \to \underbrace{A}_{\text{par}(p\leftarrow q)}\,\underbrace{B}_{\text{par}(q\leftarrow p)} \tag{11}$$

$$\underbrace{AB}_{\text{par}(q\leftarrow p)} \to \underbrace{A}_{\text{par}(q)}\,\underbrace{B}_{\text{par}(q\leftarrow p)} \tag{12}$$

$$\underbrace{I_m \otimes A_n}_{\text{par}(p)} \to \underbrace{I_p \otimes \big(I_m \otimes A_n\big)}_{\text{par}(p)} \tag{13}$$

$$\underbrace{(A_m \otimes I_n)}_{\text{par}(p\leftarrow q)} \to \underbrace{L^{mn}_m}_{\text{par}(p\leftarrow q)}\,\underbrace{(I_n \otimes A_m)L^{mn}_n}_{\text{par}(q)} \tag{14}$$

$$\underbrace{(A_m \otimes I_n)}_{\text{par}(q\leftarrow p)} \to \underbrace{L^{mn}_m (I_n \otimes A_m)}_{\text{par}(q)}\,\underbrace{L^{mn}_n}_{\text{par}(q\leftarrow p)} \tag{15}$$

$$\underbrace{L^{mn}_m}_{\text{par}(p)} \to \underbrace{\big(I_p \otimes L^{mn/p}_{m/p}\big)\big(L^{p^2}_p \otimes I_{mn/p^2}\big)\big(I_p \otimes (L^n_p \otimes I_{m/p})\big)}_{\text{par}(p)} \tag{16}$$

$$\underbrace{L^{mn}_m}_{\text{par}(q\leftarrow p)} \to \underbrace{\big(I_q \otimes (I_{p/q} \otimes L^{mn/p}_{m/p})\big)}_{\text{par}(q)}\,\underbrace{\big(L^{p^2}_p \otimes I_{mn/p^2}\big)}_{\text{par}(q\leftarrow p)}\,\underbrace{\big(I_p \otimes (L^n_p \otimes I_{m/p})\big)}_{\text{par}(p)} \tag{17}$$

$$\underbrace{L^{mn}_m}_{\text{par}(p\leftarrow q)} \to \underbrace{\big(I_p \otimes L^{mn/p}_{m/p}\big)}_{\text{par}(p)}\,\underbrace{\big(L^{p^2}_p \otimes I_{mn/p^2}\big)}_{\text{par}(p\leftarrow q)}\,\underbrace{\big(I_q \otimes (I_{p/q} \otimes L^n_p \otimes I_{m/p})\big)}_{\text{par}(q)} \tag{18}$$

4 Rescaling FFTs Using SPIRAL

The framework developed in Section 3 allows us to explore trade-offs between communication and computation. Assume a subroutine computing a DFT on p processors in parallel. Depending on the cost of communication and the speed of processors, computing on $q < p$ processors may speed up the computation. However, the initial and final data distribution on p processors is fixed by the subroutine's interface. In this situation the performance gain by computing on only q processors can easily be lost in the necessary data redistribution from p to q processors before the computation and q to p processors after the computation.

Rescaling. Using the parallelization rules in Table 1 we can systematically derive formulas that internally use less processors than at the beginning and at the end of the computation. Further, the necessary redistribution is performed as part of the communication that has to be done anyway. Thus, these formulas are candidates to speed up the whole computation without changing the subroutine interface. We call this approach *rescaling*.

Specifically, we perform downscaling (to fewer processors) together with the first occurring communication step, while upscaling is performed with the last communication step. Hence, all encapsulated communication steps profit of the reduced communication effort.

After choosing a number of processors to rescale to, there is still to decide which q of the p processors to use for calculation. On machines with non-uniform communication structure (for instance clusters of symmetric multiprocessors) this can be an important choice that strongly influences the achieved performance.

In SPIRAL, the formula rewriting is performed automatically; SPIRAL's search will find a formula, and thus a rescaling strategy that performs fastest on the given platform.

Example: Rescaled DFT. We show the rewriting process that parallelizes a DFT_{mn}, for $p \mid m, n$, across p processors and rescales it to $q \mid p$ processors for the intermediate computation steps. In SPIRAL, this derivation is done automatically. We tag DFT_{mn} for p processors and expand it using rules (1) and (10)–(12):

$$\underbrace{\mathrm{DFT}_{mn}}_{\mathrm{par}(p)} \rightarrow \underbrace{(\mathrm{DFT}_m \otimes I_n)}_{\mathrm{par}(p \leftarrow q)} \underbrace{T^{mn}_n}_{\mathrm{par}(q)} \underbrace{(I_m \otimes \mathrm{DFT}_n)}_{\mathrm{par}(q)} \underbrace{L^{mn}_m}_{\mathrm{par}(q \leftarrow p)} .$$

This introduces rescaling to q processors. Next we apply rules (9), (14), and (16)–(18) to formally parallelize:

$$\begin{aligned}
\rightarrow\ & \underbrace{\left(I_p \otimes L^{mn/p}_{m/p}\right)}_{\mathrm{par}(p)} \underbrace{\left(L^{p^2}_p \otimes I_{mn/p^2}\right)}_{\mathrm{par}(p \leftarrow q)} \underbrace{\left(I_q \otimes (I_{p/q} \otimes L^n_p \otimes I_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(I_q \otimes (I_{n/q} \otimes \mathrm{DFT}_m)\right)}_{\mathrm{par}(q)} \\
& \cdot \underbrace{\left(I_q \otimes L^{mn/q}_{m/q}\right)}_{\mathrm{par}(q)} \underbrace{\left(L^{q^2}_q \otimes I_{mn/q^2}\right)}_{\mathrm{par}(q)} \underbrace{\left(I_q \otimes (L^n_q \otimes I_{m/q})\right)}_{\mathrm{par}(q)} \underbrace{T^{mn}_n}_{\mathrm{par}(q)} \underbrace{\left(I_q \otimes (I_{m/q} \otimes \mathrm{DFT}_n)\right)}_{\mathrm{par}(q)} \\
& \cdot \underbrace{\left(I_q \otimes (I_{p/q} \otimes L^{mn/p}_{m/p})\right)}_{\mathrm{par}(q)} \underbrace{\left(L^{p^2}_p \otimes I_{mn/p^2}\right)}_{\mathrm{par}(q \leftarrow p)} \underbrace{\left(I_p \otimes (L^n_p \otimes I_{m/p})\right)}_{\mathrm{par}(p)} .
\end{aligned} \tag{19}$$

Inspection shows that the final expression is parallelized in the sense of Definition 1.

Analysis. The communication and computation cost of (19) depends on the choice of the scaling factor $k = p/q$. Table 3 summarizes the effect of scaling on packet size, number of packets, computation cost, and total data to be transmitted as function of k. In essence, scaling down keeps the overall amount of data to be transmitted practically constant while increasing the message size and the computation cost. The best choice of q depends on the relation between the speed of the processor, the communication latency, and the bandwidth.

Table 3. Effect of rescaling by $k = p/q$ on computation and communication

computation	data volume	packet size	#packets
$O(k)$	$O(1)$	$O(k^2)$	$O(1/k^2)$

5 Experimental Results

In this section we evaluate our approach. We first show that rescaling speeds up smaller DFTs. Then we show that our generated DFT programs compare favorably with FFTW.

Benchmark setup. All experiments were done with complex-to-complex double-precision 2-power FFTs. The platform is a cluster of AMD Opteron 250 CPU dual nodes running at 2.4 GHz, connected by a Mellanox InfiniBand high speed network with a theoretical peak of 10 Gb/s and 4 μs latency. All codes were compiled using the GNU C compiler 3.4.4 with the option `-O3` and linked with the mvapich 0.9.5 MPI library. Performance data is given in pseudo Mflop/s computed as $5n \log n/t$, where n is the DFT size and t the runtime in microseconds. This measure is proportional to inverse runtime and hence preserves runtime relationships. Further, it gives an indication of the absolute floating-point performance [12].

Experiment 1: Rescaling. Figures 3 (i)–(iii) show the performance impact of rescaling for the problem sizes 2^{12}, 2^{15}, and 2^{18}. We start with $p = 16$ processors and let SPIRAL generate downscaling programs for $q = 1$, 2, 4, 8, and 16 processors (16 processors implies no downscaling). We compare the performance of the original and the downscaled programs to FFTW-MPI running on 16 processors. Figures 3 (i) and (ii) show a performance peak at $q = 8$ processors. Thus, for the sizes 2^{12} and 2^{15} we gain from downscaling. Figure 3 (iii) shows that $p = 16$ processors are required for the best performance at problem size 2^{18}. For this size, the increased workload per processor overcompensates the gain in communication speed.

On the benchmark platform, rescaling speeds up only for smaller sizes. On machines with slower, higher-latency networks we saw performance gains due to rescaling for larger problem sizes.

Experiment 2: Comparison to FFTW. Figure 4 (ii) shows the speed-up of SPIRAL generated FFT programs run on 16 CPUs *without* downscaling, and *with* optimal downscaling (8 CPUs for small sizes), compared to FFTW-MPI 2.1.5 using 16 CPUs. For sizes up to 2^{17}, downscaling provides significant performance gains. For these sizes

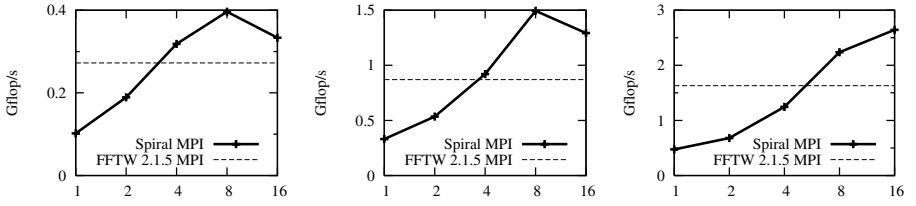


Fig. 3. Effect of downscaling from $p = 16$. The plots show, for three DFT sizes n, the best performance obtained for different scaling factors $k = p/q$. $p = 16$ and the x-axis is labeled with q. The dashed line is the performance achieved by FFTW. Higher is better.

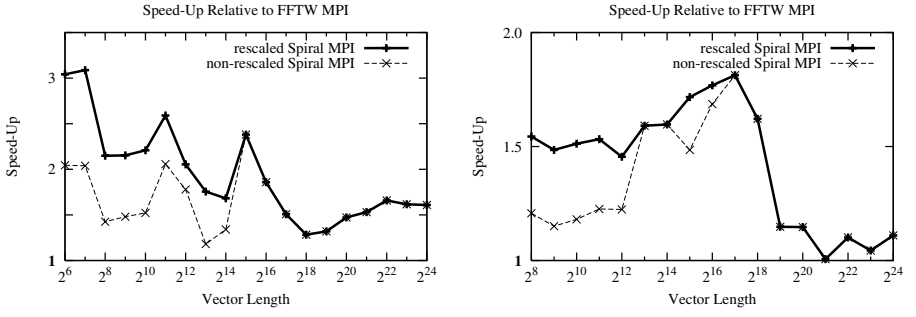


Fig. 4. Relative performance. Performance of SPIRAL generated MPI FFT programs without down-scaling (dashed), and optimally rescaled (solid), relative to FFTW-MPI 2.1.5. Higher is better.

SPIRAL generated programs are up to 80% faster than FFTW-MPI. For larger sizes, SPIRAL's performance is comparable to FFTW-MPI.

Figure 4 (i) shows the same experiment, but for 8 CPUs. The optimal downscaling found in this case is to 4 CPUs for small sizes. SPIRAL generated MPI programs are between 1.5 and 2.5 times faster than FFTW-MPI, showing higher relative speed for problems smaller than 2^{16}.

6 Conclusion

We presented a formal framework for generating efficient MPI algorithms by rewriting formulas representing FFT algorithms. We applied the framework to implement the idea of flexible rescaling and thus enable adaptation to a platform's characteristics. By including the framework into SPIRAL's infrastructure, the entire implementation and adaptation process is automated. It is worth pointing out that we used very similar approaches before to the related problems of vectorization and shared memory parallelization. In fact, all these optimizations are performed using the same infrastructure

in SPIRAL. Since our approach is formula based, it is domain-specific but can be generalized to other linear transforms.

Ongoing work aims to enable SPIRAL to optimize the runtime of the DFT including possible data redistributions. This way, the user can specify the desired data layout before and after the computation to interface with his application. As both data distribution and transform are represented on a mathematical level they can be optimized jointly, thus reducing the overhead.

References

1. A. Adelmann, A. Bonelli, W. P. Petersen, and C. W. Ueberhuber. Communication efficiency of parallel 3D FFTs. In *Proc. High Performance Computing for Computational Science (VECPAR)*, volume III, pages 901–907, 2004.
2. G. Baumgartner, A. Auer, D. E. Bernholdt, A. Bibireata, V. Choppella, D. Cociorva, X. Gao, R. J. Harrison, S. Hirata, S. Krishnamoorthy, S. Krishnan, C. Lam, Q. Lu, M. Nooijen, R. M. Pitzer, J. Ramanujam, P. Sadayappan, and A. Sibiryakov. Synthesis of high-performance parallel programs for a class of ab initio quantum chemistry models. *In [17]*, pages 276–292, 2005.
3. L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. SCALAPACK *Users' Guide*. SIAM, Philadelphia, PA, 1997.
4. N. Dershowitz and D. A. Plaisted. Rewriting. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 1, chapter 9, pages 535–610. Elsevier, 2001.
5. M. Eleftheriou, B. Fitch, A. Rayshubskiy, T. C. Ward, and R. Germain. Scalable framework for 3D FFTs on the Blue Gene/L supercomputer: Implementation and early performance measurements. *IBM Journal of Research and Development*, 49(2/3):457–464, 2005.
6. A. Faraj and X. Yuan. Automatic generation and tuning of MPI collective communication routines. In *Proc. International Conference on Supercomputing (ICS)*, pages 393–402, 2005.
7. F. Franchetti and M. Püschel. A SIMD vectorizing compiler for digital signal processing algorithms. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, pages 20–26, 2002.
8. F. Franchetti, Y. Voronenko, and M. Püschel. Loop merging for signal transforms. In *Proc. Programming Language Design and Implementation (PLDI)*, pages 315–326, 2005.
9. F. Franchetti, Y. Voronenko, and M. Püschel. FFT program generation for shared memory: SMP and multicore. In *Proc. Supercomputing (SC)*, 2006.
10. F. Franchetti, Y. Voronenko, and M. Püschel. A rewriting system for the vectorization of signal transforms. In *Proc. High Performance Computing for Computational Science (VECPAR)*, 2006. On CD-ROM.
11. M. Frigo. A fast Fourier transform compiler. In *Proc. Programming Language Design and Implementation (PLDI)*, pages 169–180, 1999.
12. M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, volume 3, pages 1381–1384. IEEE, 1998.
13. M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *In [17]*, pages 216–231, 2005.
14. G. Goumas, N. Drosinos, M. Athanasaki, and N. Koziris. Automatic parallel code generation for tiled nested loops. In *Proc. Symposium on Applied Computing (SAC)*, pages 1412–1419. ACM Press, 2004.

15. F. Gygi, E. Draeger, B. R. de Supinski, R. K. Yates, F. Franchetti, S. Kral, J. Lorenz, C. W. Ueberhuber, J. Gunnels, and J. Sexton. Large-scale first-principles molecular dynamics simulations on the Blue Gene/L platform using the Qbox code. In *Proc. Supercomputing (SC)*, page 24, 2005.
16. J. Johnson and K. Chen. A self-adapting distributed memory package for fast signal transforms. In *Proc. International Parallel and Distributed Processing Symposium (IPDPS)*, page 44a, 2004.
17. J. M. F. Moura, M. Püschel, D. Padua, and J. Dongarra, editors. *Special Issue on Program Generation, Optimization, and Platform Adaptation*, volume 93(2) of *Proceedings of the IEEE*, 2005.
18. J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. Dongarra. Performance analysis of MPI collective operations. *Cluster Computing Journal, Special Issue on Performance Modeling and Evaluation of Parallel and Distributed Systems*, 2006. Accepted for publication.
19. M. Püschel, J. M. F. Moura, J. Johnson, D. Padua, M. Veloso, B. W. Singer, J. Xiong, F. Franchetti, A. Gačić, Y. Voronenko, K. Chen, R. W. Johnson, and N. Rizzolo. SPIRAL: Code generation for DSP transforms. *In [17]*, pages 232–275, 2005.
20. Spiral web site. www.spiral.net.
21. C. Van Loan. *Computational Frameworks for the Fast Fourier Transform*, volume 10 of *Frontiers in Applied Mathematics*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1992.

Debugging Distributed Shared Memory Applications

Jeffrey Olivier, Chih-Ping Chen, and Jay Hoeflinger

Intel Corporation, Santa Clara, CA, USA
{jeffrey.v.olivier, chih-ping.chen, jay.p.hoeflinger}@intel.com

Abstract. A debugger is a crucial part of any programming system, and is especially crucial for those supporting a parallel programming paradigm, like OpenMP. A parallel, relaxed-consistency, distributed shared memory (DSM) system presents unique challenges to a debugger for several reasons: 1) the local copies of a given variable are not always consistent between distributed machines, so directly accessing the variable in the local memory by the debugger won't always work as expected; 2) if the DSM and debugger both modify page protections, they will likely interfere with each other; and 3) since a large number of SIGSEGVs occur as part of the normal operation of a DSM program, a program error producing a SIGSEGV may be missed. In this paper, we discuss these problems and propose solutions.

1 Introduction

A software distributed shared memory (DSM) system [10] allows a program to run on a set of distributed computers by simulating a hardware shared memory. This provides for what could be called "automatic message passing" to move data between the distributed systems. Programming with a DSM is significantly easier than using explicit message passing calls inserted by hand. High level parallel languages, like OpenMP [17], can be built on top of a DSM to further simplify the programming of distributed machines.

This simulation of hardware by software embedded in the program itself presents unique problems to a debugger. Debuggers typically depend upon the hardware to maintain data consistency for the program. Normally, when the debugger halts a program and queries memory, the hardware returns the appropriate value. When the debugger halts a DSM program, the consistency mechanism is halted as well.. If any memory is not up-to-date when the debugger halts the program, and the user asks the debugger to read that memory, the debugger must allow the consistency mechanism to operate in order to be guaranteed a consistent result

Additionally, if both the debugger and the DSM modify page protections as part of their normal operation – for watchpoints in the debugger and for page invalidation in the DSM – then there must be careful coordination between the two programs.

Much of the research in debugging parallel and distributed programs focuses on the scalability of the debugger [1, 12], the detection and handling of distributed breakpoints [14], and the re-execution of parallel programs [10, 15, 16]. This paper

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 833 – 844, 2006.
© Springer-Verlag Berlin Heidelberg 2006

complements these lines of research by focusing on solving the fundamental problems that break the conventional debugger functions when DSM systems are involved.

The remainder of this paper is organized as follows: Section 2 discusses the technical details of DSM systems. Section 3 discusses debuggers and the problems they encounter with DSM systems. Section 4 discusses our design and how it addresses these problems. Section 5 briefly discusses our prototype implementation.

2 Distributed Shared Memory

2.1 DSM Basics

Data referenced by more than one thread in a parallel region will be referred to in this paper as *sharable* data. The DSM system implements the consistency mechanism to be used for the sharable data. There are many choices for a consistency protocol. The simplest is sequential consistency [9]. Sequential consistency dictates that memory accesses happen in program order on each thread, and a single sequential order for the memory accesses is maintained across all threads. This is simple, but requires synchronization on each memory access, making such a code significantly slower than the equivalent parallel program on an SMP.

People have devised "relaxed consistency" mechanisms that allow the overlap of memory operations, yet still allow correct execution. Weak ordering [5] is one such relaxed consistency mechanism. Weak ordering removes the ordering constraints on all memory operations except for synchronization operations. One thread needs not see modifications made to sharable data by another thread until the two threads synchronize with each other.

DSM systems are implemented in a variety of ways, but a common way is to replicate sharable data on all systems, and to protect pages that are not fully up to date in a particular system. When the program accesses data on a protected page, a SIGSEGV signal is delivered to the program, the program gathers modifications to the page from remote systems, the modifications are applied to the local page, the protection is removed, the instruction is restarted, and this time the access succeeds.

In such a scheme, a process needs to know which remote processes have made modifications to any given page. This information is communicated at synchronization points within the code. As part of the synchronization, one process will send *write notices* for a page to another process. This allows the process that receives the write notices to keep track of which other processes must be contacted when bringing the page up-to-date. TreadMarks [8] is a well-known DSM system that uses this scheme along with a relaxed consistency protocol to maintain sharable data between processes.

2.2 The Intel DSM

Our DSM was originally based on an exclusively-licensed, enhanced version of TreadMarks. The TreadMarks code has been extensively modified. Many of its

limitations were relaxed, and it is now based on a Communications Abstraction Layer (CAL) that handles communication between distributed processes. The CAL software allows our DSM to transparently use multiple interconnection fabrics.

The consistency of data in the sharable part of the address space is provided by the execution of the DSM code. Figure 1 shows a block diagram of the consistency mechanism implemented as part of this DSM. Other software DSM systems that rely on page protections would follow a similar process.

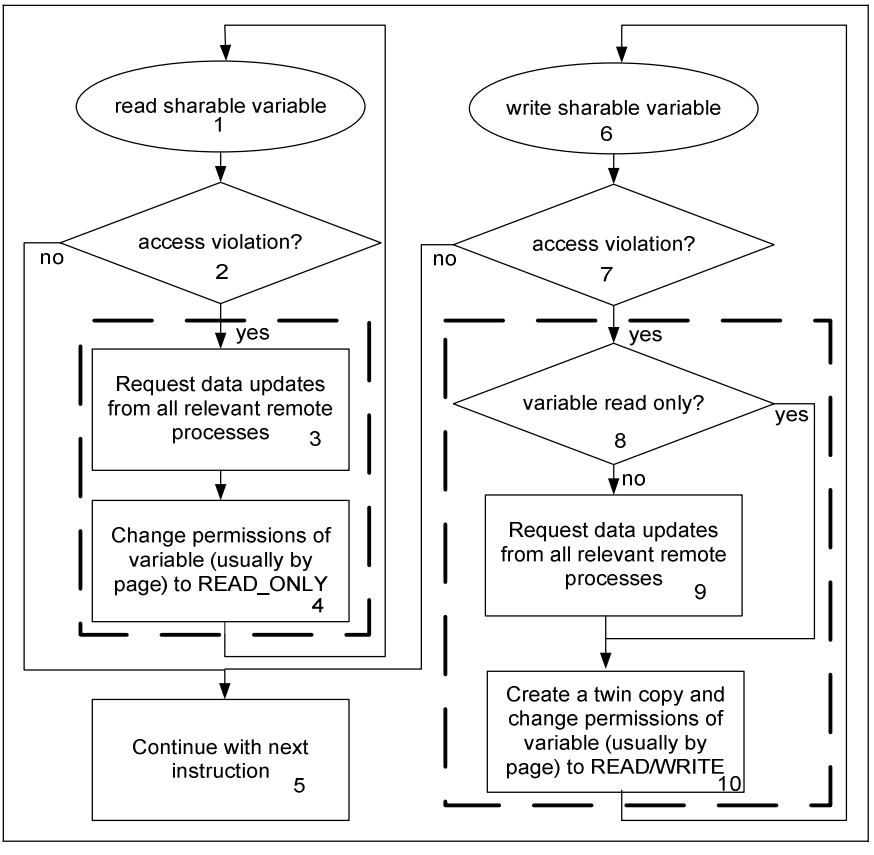


Fig. 1. Handling the reading or writing of a sharable variable within the DSM runtime library. The consistency mechanism is enclosed in dashed lines.

3 Problems in Debugging a DSM Program

3.1 Debugger Basics

Debuggers are generally used to find errors in the source code of a program. Several operations are usually available to the debugger user

- Breakpoints – A breakpoint can be set at a line or function call in the user code. The debugger can implement breakpoints by inserting an illegal instruction in the code causing the kernel to raise SIGTRAP when the code is executed. At a breakpoint, the user program is halted and the debugger begins an interactive session with the user.
- Watchpoints – Watchpoints are similar to breakpoints except they are set to watch changes to program data. One way to implement watchpoints is to protect the page on which the variable resides, causing the kernel to raise SIGSEGV when a write to the page occurs. At this point, the debugger can determine whether the access was to the variable being watched. If so, the debugger begins an interactive session with the user.
- Program state examination and manipulation – A debugger allows the user to view program state information such as register data, variable data, a source code location, an assembly language instruction, etc. Such information can often be modified or manipulated to assist in debugging the program.
- Single Stepping – Debuggers generally provide a mechanism for executing the code one line or instruction at a time. This allows the user to stop and examine state information before and after a given line of code is executed.
- Inferior calls – When the debugged process (i.e., the debuggee) is stopped, the debugger can initiate the execution of a routine defined in the debuggee. Such an execution is called an inferior call. When an inferior call returns or aborts, the debugger is responsible for restoring the execution context (such as the register contents and the execution stack) to what it was before the inferior call.

3.2 Debugging a DSM

3.2.1 Sharable Variables

When debugging an application on a hardware shared memory machine, a debugger can make the assumption that the value that is read from memory is the correct value because the underlying hardware maintains the consistency of the shared memory. Debuggers for MPI [13] can make the same assumption because there are no variables that are shared between nodes, so consistency beyond what the hardware provides is not an issue.

In DSM systems, however, the page containing a sharable variable may be invalidated at any time, indicating that the value is out-of-date. If such a variable is read from memory by the debugger without allowing the consistency mechanism to run and bring it up-to-date, then the debugger may receive a value that is different from what the program would have read at the same point in the execution.

Furthermore, each process may have a different up-to-date value depending on synchronization patterns between processes. For example, if a process writes to a sharable variable and then the debugger is stopped, that process will have a different value than any other processes for that variable. Another process would only get an up-to-date value after the next synchronization with the writing process. Looking at the values of the variable on all processes might be confusing to a naïve user.

3.2.2 DSM Signal Handling

When software DSM systems use page protections to keep sharable memory consistent between processes, they must intercept SIGSEGV signals for their coherence protocols to work. Debuggers may intercept SIGSEGV for two purposes:

1. To catch program errors.
2. To implement watchpoints.

Both of these uses conflict with signal handling in the software DSM.

4 Solving the DSM Debugging Problems

In the following sections, we propose solutions to the problems we have pointed out with debugging a DSM program. The solutions require modifications to both the debugger and the DSM, and require significant cooperation between the two mechanisms but they do not require any modifications to the application.

4.1 Debugging Interface

To correctly present information about a DSM process to the user, the debugger and the DSM need to cooperate. One way to do this is to use a *debug assistant interface* to specify a set of functions that the debugger can call to communicate with the DSM. A similar setup is used for MPI, pthreads, and UPC [2,4,6]. The interface would also stipulate using an *event function,* which the DSM system would call in order to transfer control to the debugger when some debug event occurs. We will refer to this event function as `debug_event` in the rest of this paper.

When the debugger is about to debug a DSM process, the debugger dynamically loads the shared library that implements the debug assistant interface for the DSM system. The debugger then creates an *agent* for the process. The agent serves all requests on the DSM process from the debugger making use of debugger provided callback functions to read and write data from the process and to make inferior calls. After the agent is created, the debugger sets a breakpoint on `debug_event` so that it can take control when the event function is called by the DSM system. After the breakpoint is set, the debugging activity on the DSM process can proceed.

The debugger in this modular design would implement only the generic capability to present to the user the DSM-specific information, whereas the debug assistant interface would gather that information for the debugger. If designed properly, one debug assistant interface can support *many* DSM systems. The debugger only needs to load the appropriate shared library implemented for the DSM system that the process being debugged runs on. This not only dramatically reduces the complexity of the debugger, but it also makes supporting new DSM systems more cost effective because only a new implementation of the interface needs to be written.

4.2 Safe State

4.2.1 Safe State Defined

As explained earlier, a DSM contains a memory consistency mechanism that the debugger must allow to run in order to guarantee a consistent view of memory for the user of the debugger. In order to do this, the debugger must ensure that the consistency mechanism continues to run while inspecting memory values in a program. Otherwise, the debugger has no guarantee that it is presenting the proper view of memory to the user. When the debugger halts a program at some arbitrary point, the debugger must make sure that any memory consistency operations that are in-flight are allowed to finish. A process that has no such operations outstanding and no threads in a state that will prevent any such operations from executing is said to be in a *safe state.*

Definitions. A *focus thread* in a DSM program is the thread that raises a debugger event. The *DSM runtime* (or just *runtime*) refers to the runtime library routines that enable the execution of a DSM program. The *user threads* are the threads that execute the user code. The *helper threads* handle consistency and synchronization requests between processes. These threads are part of the DSM runtime. A *safety barrier* is a state that safe threads enter, to wait until all threads are safe.

A user thread in a DSM process is stopped in a *safe state* if and only if both of the following are true:

1. The thread is currently executing the code outside the dashed line box in Figure 1. If a thread is currently executing inside the box, then the state of the thread is currently "in transition to a safe state". The state of sharable memory is unknown during the transition.
2. The thread is not holding a resource that would prevent another thread in the same process from entering or leaving the code in the dashed line box of figure 1. Failure to ensure this condition can result in deadlock if the focus thread manipulates sharable memory.

A helper thread in a DSM process is always in a safe state, if it is running. If a helper thread is stopped, it must meet both of the following conditions to be safe:

1. It is part of the focus thread's process. If a helper thread is not part of the focus process then it must continue to run in order to service requests from the focus process.
2. It meets condition 2 of the user thread's safe state criteria.

A DSM process is in a safe state if and only if all of the threads in the process are in a safe state. Only the focus thread can leave the safe state before the user continues the application. A DSM application is in a safe state if and only if all of its processes are in a safe state. We say a thread/process/application is safe if and only if it is in a safe state; otherwise, it is unsafe.

In general, all debugger operations that have the potential to access sharable memory require a safe process. These operations include sharable memory read, sharable memory write, and single-stepping an instruction. Failure to do so may result in a deadlock.

4.2.2 Safe State Transition

If the debugger must do an operation **O** that requires a safe application, it must cause all threads to transition to a safe state. The following process must be followed to do the transition:

1. The debugger informs the runtime to prepare to get all threads into a safe state.
2. The debugger obtains the set **U** of unsafe user threads via the interface. It also obtains the set **S** of safe user threads. The debugger stops the threads in **S**. The threads in **U** and the helper threads will continue to run. Each thread in **U** executes until it becomes safe, at which time it is forced into a safety barrier.
3. The debugger receives an event from the runtime via the event function when all threads in **U** are saf e.
4. The debugger stops all user threads, and does the operation **O** on the focus thread.
5. When **O** completes, the debugger informs the runtime that the operation has completed.

Note that the helper threads are resumed in Step 2 and are still running in Step 4 when the debugger does **O**.

4.2.3 Reading and Writing Sharable Variables

If the operation **O** to be done by the debugger is a read or a write of a sharable variable **V** at memory address **M**, the debugger must go through the safe state transition as described above. In Step 4 of the safe state transition, it must bring the page upon which the variable resides up-to-date. One way for the debugger to update **M** is to make an inferior call in the debuggee to leverage the memory consistency protocol in the DSM. That is, the inferior call does the memory access inside the application,, triggering the software memory consistency mechanism.

For a read from variable **V**, the debugger makes an inferior call on a routine that does a memory copy (e.g. *memcpy*) of the size of **V** from **M** to a buffer **B**. If **V** is invalid (i.e., the memory page on which **M** resides is read protected), the memory copy triggers a SIGSEGV and the SIGSEGV handler in the DSM would update **V**. After the inferior call returns, the debugger can get the up-to-date value of **V** from **B**.

For a write to variable **V**, the debugger writes the new value for the variable **V** to the buffer **B**. Then, the debugger makes an inferior call on a routine that does a memory copy from **B** to the memory address **M**. If **M** is write protected, the copy operation raises a SIGSEGV, triggering the DSM memory consistency protocol to record the change made to **V**.

4.3 DSM and Debugger Coordination

4.3.1 SIGSEGV Coordination

To allow for coordination between the debugger and the DSM, the debugger would set a breakpoint in the `debug_event` function in the DSM and pass all SIGSEGV signals on to the DSM application by default. If the DSM encounters a SIGSEGV signal that it can't handle, it will inform the debugger by calling `debug_event` (with arguments indicating a SIGSEGV) There are two cases where this might happen:

1. A programming error results in a SIGSEGV outside of sharable memory.
2. The debugger has set the permissions on a page such that the access would have faulted if the DSM was not intercepting the SIGSEGV signal. If the DSM protections also would have caused a fault, the DSM should always do its work before calling `debug_event` to signal the debugger.

4.3.2 Page Protection Coordination

The DSM needs to know what the debugger protections are for a given page. From the debugger side, this can be done in one of two ways:

1. The page protection can be set through an inferior call which can be handled through an interface routine. In this case the DSM is responsible for changing protections on a page and storing the debugger protections.
2. The debugger can query the current DSM protections of the page before changing the protections and then inform the DSM via an interface routine that the debugger protections have changed.

In either case, to ensure that all SIGSEGV signals are caught appropriately, the protections must be set as follows:

page protection = max_protection(DSM_protections, debugger_protections)

4.3.3 Watchpoints

Given the SIGSEGV and protection coordination schemes, it is simple to see how SIGSEGV watchpoints might be implemented. Figure 2 and Figure 3 show pseudo code for one possible implementation. Figure 2 contains routines implemented by the debugger whereas Figure 3 contains routines implemented by the DSM.

In order to show that the algorithm for watchpoints works without compromising the consistency of the memory model of the DSM, it is sufficient to consider the four possible scenarios as illustrated below:

1. A SIGSEGV is raised on a page that isn't protected by either the runtime or the debugger. That is, the SIGSEGV is purely an error in the user code. In this case, the statement labeled `NO_SHAR` in Figure 3 is executed in the runtime. As a result of that, the debugger takes control. Since the debugger is not watching the page, it will execute the code labeled `PROG_ERR` in Figure 2 reporting an error to the user.

```
void
set_watchpoint(watchpoint_id,address,size, protections)
{

store_watchpoint_in_table(watchpoint_id,address,size,permi
ssions);
  set_protections_in_DSM(page_start(address),protections);
}
void handle_SEGV_in_debugger(addr,ip)
{……

if(!on_watched_page(addr)||is_single_stepping_breakpoint_s
et)
     {
PROG_ERR:  This is a real error.  Report to user.
     }
     else
     {
WPT_HIT: // Prepare to single step
         set_protections_in_DSM(page_start(addr), none);
         // Usage of ip+1 means stop at next instruction
         set_single_stepping_breakpoint(ip+1);
         is_single_stepping = TRUE;
         resume the execution of the debuggee;
         wait for the debuggee to stop;
         single_step_succeeds = handle_event();
         //  Revert  back  to  the  state  before  single
stepping.
         remove_single_stepping_breakpoint(ip+1);
         is_single_stepping = FALSE;
         protection = get_debugger_protection(page_start);
         set_protections_in_DSM(page_start(addr),
protection);
         if (single_step_succeeds) {
             if (watchpoint_hit(addr)) {
                 inform user of watchpoint hit;
             } else {
                 Resume_debuggee();
             }
         }
         //if  there  is  an  error  single  stepping,  it  will
be
         //reported in the statement labeled PROG_ERR.
     }
}
```

Fig. 2. Debugger routines to implement DSM watchpoints for debuggers that use page protections to implement watchpoints. The routine `set_watchpoint` is called when a user sets a watchpoint on a variable. The routine `handle_SEGV_in_debugger` is called when a `debug_event` call indicates a SIGSEGV.

2. A SIGSEGV is raised on a page that is protected by the runtime but not by the debugger. That is, the page is sharable in the DSM but there is no watchpoint on it. In this case, the block of code labeled `DSM_SEGV` in `handle_SEGV_in_DSM` in Figure 3 is executed, setting the right protection for the page to prepare for the re-execution of the instruction that causes the SIGSEGV and brings the contents of the page up-to-date. The offending

instruction will be re-executed by the operating system when handle_SEGV_in_DSM returns. Note that the conditional labeled DBG_SEGV is false in this case, thus the debugger never takes control.

```
void set_protections_in_DSM(page_start, protection)
{
     page_struct = get_page_struct(page_start);
     if (page_struct != NULL) {
SHAR:  page_struct->debugger_protection = protection;
       set_page_protection(page_struct,
          max_protection(protection,page_struct-
>DSM_protection);
     } else { //Just change the protections on the page
       //to the debugger protection.
PRIV:  protect_page(start_addr, protection);
     }
}
void handle_SEGV_in_DSM()
{
    instr = the instruction at the current program counter;
    offending_address = the address of the memory accessed by
instr;
    if ((page  = get_page_struct(offending_address) == NULL) {
        //the page is not a sharable page
NO_SHAR:debug_event(SEGV, offending_address, instr);
    } else {
DSM_SEGV:
       if (memory_access_caused_fault(instr,
                              page_struct->DSM_protection){
          //changes DSM_protection field
          handle_fault(instr,page_struct);
          set_page_protection(page_struct,
                max_protection(page_struct-
>debugger_protection,
                          page_struct->DSM_protection);
          //page_struct->debugger_protection must default to
NONE
       }
DBG_SEGV:
       if (memory_access_caused_fault(instr,
                      page_struct->debugger_protection) {
          debug_event(SEGV, offending_address, instr);
       }
}
```

Fig. 3. DSM routines to implement DSM watchpoints for debuggers that use page protections to implement watchpoints. The routine set_protections_in_DSM is called when the debugger needs to change protections on a sharable page. The routine handle_SEGV_in_DSM is the SIGSEGV handler.

3. A SIGSEGV is raised on a page that is protected by the debugger but not by the runtime. This indicates that the page has a watchpoint on it yet the DSM doesn't care about the SIGSEGV. If the page is not in sharable memory, the statement labeled NO_SHAR in Figure 3 will be executed, triggering a debug_event call. If the page is sharable, the conditional

labeled `DBG_SEGV` in `handle_SEGV_in_DSM` in Figure 3 is true and the debugger takes control. Then, the debugger would process the watchpoint as usual with the block of code labeled `WPT_HIT` in the routine `handle_SEGV_in_debugger` in Figure 2. Note that the routine `set_protections_in_DSM` in Figure 3 handles both private and sharable page protection settings. The difference is that the DSM doesn't need to store debugger protections on private pages.

4. A SIGSEGV is raised on a page that is protected both by the runtime and the debugger. This is basically a combination of 2 and 3 above, with one difference. In 2, the instruction is re-executed by the operating system, whereas here the instruction is re-executed by the debugger code that handles the watchpoint hit.

5 Implementation

OpenMP provides a productive environment for writing parallel codes for hardware shared memory machines. It is generally easier to parallelize an existing serial code with OpenMP than to rewrite it for a message passing system like MPI.

Due to its wide acceptance and the fact that it uses a relaxed consistency shared memory model, it is natural to consider running OpenMP programs on a DSM system. Cluster OpenMP [7] is an add-on feature of the Intel® compilers, version 9.1 and later. It implements OpenMP on top of our DSM system.

An implementation of the design described in Section 4 is prototyped in the Intel® Debugger (IDB) for use with the Cluster OpenMP runtime library. When IDB is started, it identifies the processes in a Cluster OpenMP job using a mechanism similar to that used for MPI. IDB brings the job under control by attaching one debugger to each process in the job [3]. In order to do this the debugger loads a library that implements an interface between the debugger and the runtime library. The debugger provides several callbacks for reading and writing to the process memory, retrieving symbol addresses, and performing inferior calls.

Our implementation relies heavily on the use of inferior calls and, therefore, does not yet support core file debugging. For example, for safe state transition illustrated in section 4.2, each query through the interface will ultimately query the runtime through an inferior call. The runtime supplies a global buffer that the interface library can read and write on behalf of the debugger.

6 Conclusion

Debugger design for a DSM system must be done carefully. If both the debugger and the DSM alter memory protections, then they must coordinate changes with each other. Also, since memory consistency is maintained by code in the executable, that code must be allowed to run when the debugger manipulates the program's sharable variables. Our design solves fundamental problems that conventional debuggers encounter when debugging DSM applications. The debugger and DSM cooperate via a debugger interface. It introduces the novel ideas of safe state transition, SIGSEGV coordination, page protection coordination, and a watchpoint handling algorithm.

References

1. S. M. Balle, B. R. Brett, C. Chen, and D. LaFrance-Linden. Extending a traditional debugger to debug massively parallel applications. In Journal of Parallel and Distributed Computing 64(5): 617-628, 2004.
2. W.W. Carlson, J.M. Draper, D.E. Culler, K. Yelick, E. Brooks, K. Warren,: Introduction to UPC and Language Specification. Technical Report CCS-TR-99-157, Institute for Defense Analysis, Center for Computer Sciences, Bowie, Maryland, 1999.
3. C. Chen. The Parallel Debugging Architecture in the Intel® Debugger. In Proceedings of the 7th International Conference, Parallel Computing Technologies 2003, volume 2763 if LNCS, pages 444-451. Springer-Verlag Berlin Heidelberg 2003.
4. J. Cownie and W. Gropp. A standard interface for debugger access to message queue information in MPI. In 6th European PVM/MPI Users' Group Meeting, volume 1697 of LNCS, pages 51-58. Springer-Verlag, 1999.
5. M. Dubois, C. Scheurich and F. A. Briggs, Memory Access Buffering in Multiprocessors, In Proceedings of the Thirteenth Annual International Symposium on Computer Architecture 14, 2, pages 434-442, June 1986.
6. Etnus LLC: TotalView Reference Guide, Version 6.0. Etnus LLC, 2002.
7. Intel Corporation: Cluster OpenMP User's Guide, Version 9.1, Intel Corporation, 2005-2006.
8. P. Keleher, A. L. Cox, and W. Zwaenepoel. Lazy release consistency for software distributed shared memory. In Proceedings of the 19th Annual International Symposium on Computer Architecture, pages 13--21, May 1992.
9. L. Lamport: How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs. In IEEE Trans. Computers 28(9): 690-691, 1979.
10. K. Li and P. Hudak. Memory Coherence in Shared Virtual Memory Systems . Proceedings of the 5th ACM Symposium on Principles of Distributed Computing, 1989.
11. T. J. LeBlanc and J. M. Mellor-Crummey. Debugging parallel programs with instant replay. In IEEE Transaction on Computers, 36(4):471-482, 1987.
12. S. S. Lumetta and D. E. Culler. The Mantis Parallel Debugger. In Proceedings of SPDT'96: SIGMETRICS Symposium on Parallel and Distributed Tools, 1996.
13. Message Passing Interface Forum. MPI: A Message Passing Interface Standard. Version 1.1, June 1995.
14. B. P. Miller and J. Choi. Breakpoints and Halting in Distributed Programs. In Proceedings of the 8th International Conference on Distributed Computing Systems (ICDCS), 1988.
15. N. Mittal and V. K. Garg. Debugging Distributed Programs Using Controlled Re-execution. In Proceedings of the 19th Annual ACM Symposium on Principles of Distributed Computing (PODC), 2000.
16. R. H. B. Netzer. Optimal tracing and replay for debugging shared-memory parallel programs. In Proceedings of ACM/ONR Workshop on Parallel and Distributed Debugging, pages 1--11, San Diego, California, May 1993..
17. OpenMP Architecture Review Board: OpenMP Application Program Interface, Version 2.5. OpenMP Architecture Review Board. 2005..

Implications of Memory Performance for Highly Efficient Supercomputing of Scientific Applications

Akihiro Musa[1,2], Hiroyuki Takizawa[1], Koki Okabe[1], Takashi Soga[3], and Hiroaki Kobayashi[1]

[1] Tohoku University, Sendai, 980-6025, Japan
musa@sc.isc.tohoku.ac.jp, {tacky, okabe, koba}@isc.tohoku.ac.jp
[2] NEC Corporation, Tokyo, 108-8001, Japan
[3] NEC System Tecnologies,Osaka, 540-8551, Japan
soga-txa@necst.nec.co.jp

Abstract. This paper examines the memory performance of the vector-parallel and scalar-parallel computing platforms across five applications of three scientific areas; electromagnetic analysis, CFD/heat analysis, and seismology. Our evaluation results show that the vector platforms can achieve the high computational efficiency and hence significantly outperform the scalar platforms in the areas of these applications. We did exhaustive experiments and quantitatively evaluated representative scalar and vector platforms using real applications from the viewpoint of the system designers and developers. These results demonstrate that the ratio of memory bandwidth to floating-point operation rate needs to reach 4-bytes/flop to preserve the computational performance with hiding the memory access latencies by pipelined vector operations in the vector platforms. We also confirm that the enough number of memory banks to handle stride memory accesses leads to an increase in the execution efficiency. On the scalar platforms, the cache hit rate needs to be almost 100% to achieve the high computational efficiency.

1 Introduction

Supercomputing systems are categorized into vector-parallel and scalar-parallel supercomputers. The mainstream of supercomputers has been dominated by the commodity-based scalar-parallel platforms. However, the growing gap between sustained and peak performance for scientific applications on the scalar-parallel platforms has become remarkably exposed year by year in high performance computing. On the vector-parallel platforms, the Earth Simulator, which is the largest vector-parallel computing system in the world, has achieved a sustained performance (efficiency) of 26.58 Tflops (64.9%) for a global atmospheric simulation and 16.4 Tflops (50.0%) for a turbulence simulation [1,2]. Recently, Oliker et al have compared the application performance of the scalar-parallel platforms with that of the vector-parallel platforms [3,4]. These researches show that the vector-parallel platforms have the potential for excellent performance on

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 845–858, 2006.
© Springer-Verlag Berlin Heidelberg 2006

scientific applications. On the other hand, the performance characteristics of the vector platforms have been researched since 1980's. Fatoohi show that the important factors of the sustained performance on the vector platforms are the average vector length, the ratio of floating point operations to memory references and the memory strides [5,6]. In addition, Shan and Strohmaier have investigated the memory performance characteristics of a modern vector-parallel platform: Cray X1, and show the average vector length and the memory bank conflicts have a great impact on the sustained performance [7].

The sustained performance of supercomputers strongly depends on their memory systems. The vector parallel platforms employ the interleaved memory systems to improve the memory access performance, while the scalar parallel platforms use the hierarchical cache memory systems. This paper first evaluates supercomputers from the viewpoint of memory access performance using real applications, and then clarifies the requirements for high performance computing of scientific applications. The contribution of this paper is to quantitatively discuss the effects of "byte/flop (B/FLOP)" and the number of memory banks of the vector systems on the sustained system performance when executing real applications in the fields of leading computational science. In particular, although a 4 B/FLOP is empirically believed as the golden rule for high-performance vector platforms, there is no quantitative discussion on how this metric affects the performance in executing real applications.

The rest of the paper is organized as follows. Sections 2 and 3 briefly describe the evaluated platforms and scientific applications we used, respectively. In Section 4, performance of the memory systems on these applications is analyzed. Finally, Sections 5 summarizes our results.

2 Architectural Characteristics of the Platforms

We compare scientific applications performance of vector-parallel systems: NEC SX-7 and SX-7C, with scalar parallel platforms equipped with the Intel Itanium2: NEC TX7 and SGI Altix3700. The SX-7 and SX-7C, which are installed at Tohoku University in Japan, are representative systems in modern vector-parallel systems and their architectures are similar to the Earth Simulator. Table 1 summarizes the architectural characteristics of the four platforms. The

Table 1. Architectural Summary of the Platforms

Platform	CPUs/Node	Clock Freq. (MHz)	Per CPU			Processor Types
			Peak Perf. (GFLOPS)	Memory BW (GB/s)	L3 Cache (MB)	
SX-7	32	1104	8.83	35.3	-	Custom processor
SX-7C	8	2000	16.0	64.0	-	Custom processor
TX7/i9510	32	1600	6.4	6.4	9	Intel Itanium2
Altix3700	64	1600	6.4	6.4	6	Intel Itanium2

memory bandwidths of the SX-7 and SX-7C are 5.5 and 10 times higher than those of the TX-7 and Altix, respectively. On the other hand, the scalar platforms employ large on-chip caches to cover the lower memory bandwidth.

2.1 Vector-Parallel Platforms: SX-7 and SX-7C

The SX-7 and SX-7C are shared-memory vector-parallel systems. A node of the SX-7 contains 32 processors for peak performance of 282.5 Gflops with 256GB main memory [8], and that of the SX-7C contains eight processors for peak performance of 128 Gflops with 128GB main memory. The SX-7 and SX-7C run SUPER-UX (R14.1, and R15.1, respectively) a 64-bit UNIX operating system based on System V R3 with the BSD features. FORTRAN compiler, FORTRAN90/SX R.316, supports ANSI/ISO Fortran95 in addition to functions of automatic vectorization and automatic parallelization.

Their processor has a vector operation unit and a 4-way superscalar operation unit. The SX-7's vector operation unit contains four vector pipes (Logical, Add/Shift, Multiply, Divide) with 144KB vector registers, and achieves a peak performance of 8.83 Gflops. Similarly, the SX-7C's vector operation unit contains four vector pipes (Logical, Add/Shift, Multiply, Divide/SQRT) with 144KB vector registers, and achieves a peak performance of 16 Gflops. The 4-way superscalar operation units of the SX-7 and SX-7C achieve peak performances of 1.1 Gflops and 2 Gflops, respectively. The memory bandwidths of the SX-7 and SX-7C are 35.3 GB/s with DDR-SDRAMs and 64 GB/s with DDR2-SDRAMs, respectively. The memory bandwidth per flops of the SX-7 and SX-7C is 4 B/FLOP.

The SX-7 and SX-7C use an interleaved memory system for the main memory. The interleaved memory system organizes memory chips in banks to access multiple words at a time. The memory latency of the second memory access or later are hidden in the interleaved memory system. Here, the certain number of memory banks named *minimum number of banks* is needed to hide the memory latency (bank cycle time) by sequential memory access. The minimum number of banks N_m is

$$N_m = B_w \times B_c / D$$

where B_w is memory bandwidth (GB/s) per processor, B_c is bank cycle time of memory (ns), and D is the size of a word for floating-point data: 8 bytes. Hence, N_m of the SX-7 is 353 banks per processor, and that of the SX-7C is 512 banks per processor. The SX-7 contains 512 banks per processor, 16,384 banks per node, and the SX-7C contains 512 banks per processor, 4,096 banks per node. Thus, the SX-7 has a margin of 159 banks; however, the SX-7C does not have a margin. We will examine how these margins affect the performance through the experiments using real applications.

In addition, the SX-7 and SX-7C can hide the memory access times by pipelined vector operations when a vector operation ratio (VOR) and an average vector length (AVL) of applications are large. This has a beneficial effect on high performance of the SX-7 and SX-7C. Figure 1 shows a pipeline

diagram of the following loop. The load times of C and D are hidden by the vector operations of B * C and V + D.

```
DO I = 1, N
   A(I) = B(I) * C(I) + D(I)
END DO
```

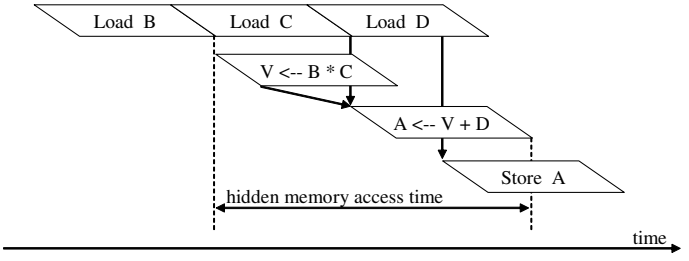


Fig. 1. Pipeline diagram: memory access times are hidden by pipeline operations. Each parallelogram shows load/store pipelines, multi pipeline, and add pipeline.

2.2 Scalar-Parallel Platforms: TX7/i9510 and Altix3700

The TX7/i9510 and Altix3700 are ccNUMA (cache coherent Non Uniform Memory Access) systems. A node of the TX7/i9510 contains eight cells and crossbar network modules [9]. Each cell contains four Intel Itanium2 processors and a 32GB main memory, which are interconnected by a 6.4GB/s bus. A computational building block of the Altix3700 consists of four Intel Itanium2 processors, main memory, and two controller ASICs called the SHUB, which connect the processors and memory at 6.4GB/s bandwidth. The Altix interconnect is called the NUMAlink, custom network in a fat-tree topology. The Itanium2 has 3-tier on-chip data caches consisting of 32KB of L1, 256KB of L2, and 6MB (Altix) /9MB (TX7) of L3. The Itanium2 does not use the L1 data cache to store floating-point data.

The performance of memory system depends on the cache system. In the case of the TX-7, the memory access time is 20 times or more as long as the L3 cache access time. Therefore, when a cache hit rate is low, a memory access time becomes dominant in the total processing time; the computational efficiency on the cache based platforms gets worse accordingly.

The TX7 and Altix run 64-bit Linux (RedHat AS2.1 and SGI Advanced Linux Environment, respectively), and supports Fortran95 (NEC R4.3) with optimization functions and parallel processing functions specialized for the Itanium2.

3 Scientific Applications

Five leading applications from three areas in scientific computing are used to compare the sustained performance of the SX-7 and SX-7C with that of the

TX7 and Altix3700. The applications have been developed by researchers of Tohoku University and are representative of each research area. Table 2 shows the summary of the applications whose methods are standard in individual areas.

3.1 GPR Simulation

This code is for a simulation of an array antenna SAR-GPR (Synthetic Aperture Radar - Ground Penetrating Radar), which detects anti personnel mines in shallow subsurface [10]. The GPR simulation evaluates performance of the SAR-GPR in detection of buried mines. The simulation method is based on the three dimensional FDTD (Finite Difference Time Domain) method with Berenger's PML (Perfectly matched layer) [11]. The simulation space consists of two regions; air-space and subsurface space with PML of 10 layers. The performance of this code is primarily determined by the electromagnetic filed calculation processes. The basic computational structure of the processes consists of triple-nested loops accessing the memory at non-stride 1 addresses; the ratio of its calculation cost to the total is 80%. The length of the innermost loop is over 500.

3.2 APFA Simulation

Radiation patterns of an Anti-Podal Fermi Antenna (APFA) are simulated to design high gain antennas [12]. The simulation consists of two sections, a calculation of the electromagnetic filed around an APFA using the FDTD method with Berenger's PML, and an analysis of the radiation patterns using the Fourier transform. The performance of the simulation is primarily determined by calculations of the radiation patterns; the ratio of its calculation cost to the total is 99%. The computational structure of the calculations is triple-nested loops; the innermost loop is a stride 1 loop, and its length is 255. On Itanium2, the innermost loop is executed on the caches. The ratio of floating-point operations to memory references in the innermost loop is 2.25. Therefore, this code is computational-intensive, and the performance of the code is not dominated by memory references.

Table 2. Summary of scientific applications

Areas	Names and Descriptions	Methods	Subdivisions
Electromagnetic Analysis	GPR simulation: Simulation of Array Antenna Ground Penetrating Radar	FDTD	50×750×750
Electromagnetic Analysis	APFA simulation: Simulation of Anti-Podal Fermi Antenna	FDTD	612×105×505
CFD/Heat Analysis	PRF simulation: Simulation of Premixed Reactive Flow in Combustion	DNS	513×513
CFD/Heat Analysis	SFHT simulation: Simulation of Separated Flow and Heat Transfer	SMAC	711×91×221
Seismology	PBM simulation: Simulation of Plate Boundary Model on Seismic Slow Slip	friction law	32400×32400

3.3 PRF Simulation

This code provides direct numerical simulations of two-dimensional Premixed Reactive Flow (PRF) in combustion for design of engines of planes [13]. The simulation uses the 6th-order compact finite difference scheme and the 3rd-order Runge-Kutta method for time advancement. The performance of the code is primarily determined by calculations of derivations of physical equations; the ratio of its calculation cost to the total is 50%, and the rest of the cost has been distributed to various routines. The calculations have doubly nested loops; the loop of x-derivations induces stride 1 memory accesses, and the loop of y-derivations induces non-stride 1 memory accesses. The length of each innermost loop is 513.

3.4 SFHT Simulation

This simulation code realizes direct numerical simulations of three-dimensional laminar Separated Flow and Heat Transfer (SFHT) on surfaces of a plane [14]. The finite-difference forms are the 5th-order upwind difference scheme for space derivatives and the Crank-Nicholson method for a time derivative. The resulting finite-difference equations are solved using the SMAC method. The performance of the code is primarily determined by calculations of the predictor-corrector methods; the ratio of its calculation cost to the total is 67%, and the rest of the cost has been distributed to various routines. The calculations have triple-nested loops; the innermost loop needs stride 1 memory accesses, and its length is 349.

3.5 PBM Simulation

This code uses the three-dimensional numerical Plate Boundary Models (PBM) to explain an observed variation in propagation speed of postseismic slip [15]. This is a quasi-static simulation in an elastic half-space including a rate- and state-dependent friction. The performance of the simulation is primarily determined by a process of thrust stress with the Green function; the ratio of its calculation cost to the total is 99%. The computational structure of the process is a doubly nested loop which calculates a product of matrices, the innermost loop results in stride 1 memory accesses, and its length is 32400.

4 Experimental Results and Discussion

The experiments conducted in this work measure the performance of the original source codes, which have been developed for SX-7, with optimizations of the compilers; compiler's options are high-level optimizations (SX: -C hopt, TX, Altix: -O3) and inlining subroutines. We used NEC compiler for the Intel Iitanium2 on the TX7 and Altix, to evaluate the performance under the same level optimizations. On SX-7 and SX-7C, the five applications are vectorized by the compiler. The applications are automatically parallelized by the compiler on four studied platforms. All the performance statistics of four studied platforms were obtained using the NEC compiler option *ftrace*.

Table 3. Characterizations of the five applications on the evaluated platforms

	SX-7/7C		TX/i9510		Altix3700		
	VOR	AVL	L2	L3	L2	L3	PR
GPR	99.7%	245.1	70.6%	40.2%	71.0%	42.6%	98%
APFA	99.5%	255.5	99.9%	26.7%	99.9%	26.8%	99%
PRF	99.3%	179.0	89.6%	78.9%	89.5%	79.9%	93%
SFHT	99.4%	192.9	92.4%	21.8%	92.4%	21.7%	98%
PBM	99.5%	255.5	88.7%	54.8%	88.9%	63.2%	98%

4.1 Characterizations of Applications

To characterize computation behavior we show a vector operation ratio (VOR) and an average vector length (AVL) on the vector platforms, the L2 and L3 cache hit rates on Itanium2, and a parallel ratio (PR) of thread-level parallelism, which is the fraction of the code executed in parallel. As Table 3 shows, these five applications are highly vectorized and parallelized, and the L2 cache hit rates range from 70% to 99.9% according to their irregularity in memory accesses.

4.2 Efficiency of Applications on Four platforms

The overall performance comparison of the four platforms for the five applications is shown in Figure 2. The vector platforms achieve the high computational efficiency of over 40% and the higher sustained performance across all of the applications. The scalar platforms show that the computational efficiency is less than 14% across all of the applications. We will discuss the effects of the performance of memory access on computational efficiency of the vector and scalar platforms later in Sections 4.3 and 4.4.

The SX-7 and SX-7C can hide the memory access times by the interleaved memory system and the pipelined vector operations because VOR and AVL of the five applications are large. The memory access times not hidden by overlapping

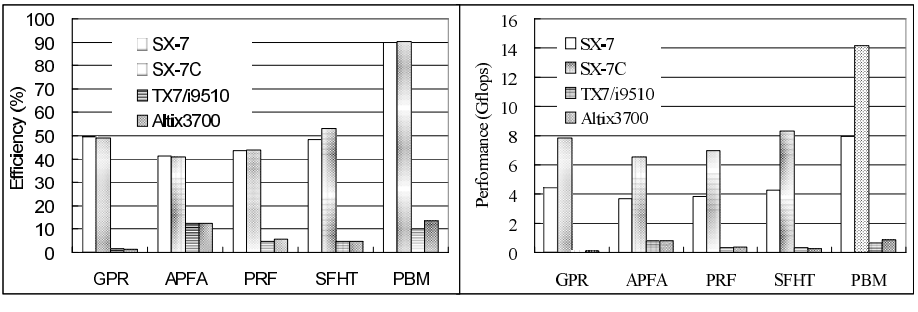


Fig. 2. Overview of performance for the five applications on one processor, computational efficiency and sustained performance

calculations of the four platforms are showed in Table 4. The non-hidden memory access time of the vector platforms are much shorter than that of the scalar platforms. In particular, the non-hidden memory access time of PBM simulation on the SX-7 is 5 seconds. However, the sum of calculated floating-point data is a 13.5 trillion; 108 terabytes in the PBM. Hiding memory access latency by pipelined vector operations works best for the PBM, because the PBM simulation has the longest loop length (32400) among the five applications and further sequentially accesses memory. On the cache based platforms, the TX7 and Altix, the non-hidden memory access times are 600+ times longer than that of the SX-7. In the longer loops of larger simulations, the vector platforms are more advantageous in the performance.

Table 4. Memory access time (Sec) and ratio of SX-7 for the five applications on one processor

	GPR		APFA		PRF		SFHT		PBM	
Platform	Sec	Ratio	Sec	Ratio	Sec	Ratio	Sec	Ratio	Sec	Ratio
SX-7	171	1	17	1	40	1	81	1	5	1
SX-7C	90	0.5	3	0.2	21	0.5	32	0.4	6	1.2
TX7/i9510	23399	137	568	34	2170	54	3674	45	5652	1082
Altix	26319	154	612	37	1745	43	3910	48	3323	636

Figure 3 shows the ratio of non-hidden memory access time to processing time for the five applications on one processor of the four platforms. The processing time of the scalar platforms consists mostly of the non-hidden memory access time. The APFA simulation achieving a 99.9% cache hit rate shows the smallest ratio of non-hidden memory access time among the five applications on the scalar platforms; however the ratio is still over 20%. The memory systems of the vector platforms are more effective for science applications than that of the scalar platforms.

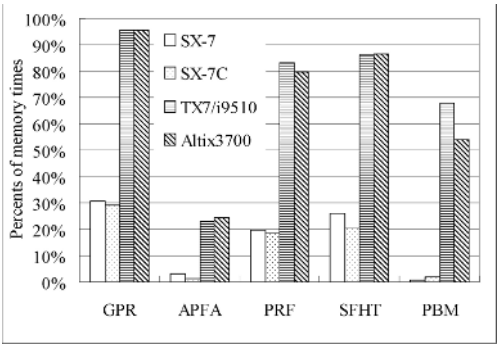


Fig. 3. The ratio of memory access time to processing time on one processor

Figure 4 shows the speedup ratio in 32 processors of our studied platforms for the five applications. The speedup ratio of the PRF simulation is the lowest among the five applications on each platform, because the parallel ratio (PR) of the PRF is 93% and lowest. The SX-7 outperforms the other platforms across all of the applications. The TX7 and Altix utilize the same processors; however, the Altix scalability is higher than the TX7. This is due to a lot of access contentions on the TX-7 bus with the lower bandwidth.

4.3 Discussion on the Memory Performance of SX-7 and SX-7C

We use the PRF and SFHT simulations to examine the non-overlapped memory access latencies when changing of the number of banks per processor on the SX-7. The performance of the PRF is dominated by memory references, because the memory access has a 4104-byte stride, and the ratio of operations to memory references is 0.7. In the SFHT, the memory access needs a 16-byte stride, and the ratio of operations to memory references is 1.0. Thus, the PFR is more memory-intensive than the SFHT. The experimental results shown in Figure 5 indicate that the non-hidden memory access time increases as the number of banks decreases. When one processor of the SX-7 has 16K banks, the non-hidden memory access times of the PRF and SFHT are 40 and 81 seconds, respectively. When one processor has 0.5K banks, the non-hidden memory access times of the PRF and SFHT are 113 and 131 seconds, respectively. The non-hidden memory access time of PFR increases faster than that of SFHT. This is because the strides of memory access of PFR are longer than that of SFHT, and needs more banks to reduce the memory access time.

In general, a lot of scientific applications need non-stride 1 memory accesses, and therefore the number of banks per processor should be more than the minimum number of banks to keep the higher computational efficiency. To evaluate the effect of the number of memory banks on the performance, we compare the performance of 8-parallel GPR simulation between the SX-7 and SX-7C. The memory access of the GPR has a 576-byte stride. Figure 6 shows that the efficiency of the SX-7 is 1.7 times higher than that of the SX-7C in the case eight processors. Although the peak performance of the SX-7C is 1.8 times faster

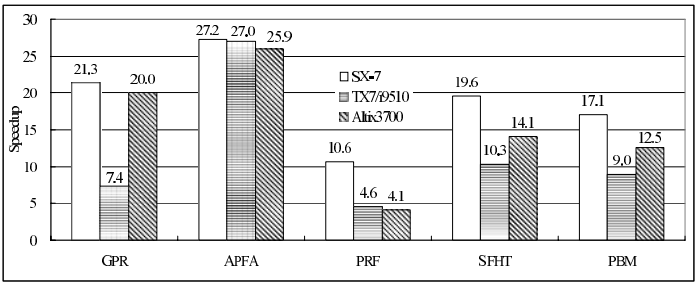


Fig. 4. Speedup ratio in 32 processors for the five applications

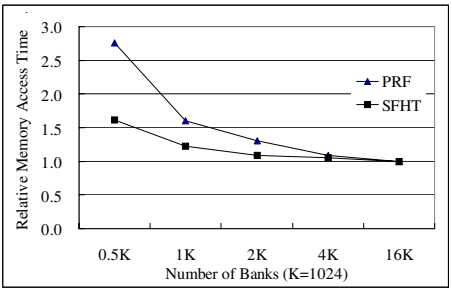


Fig. 5. Relative non-hidden memory access time of the number of banks with base of 16k banks on the SX-7

than that of the SX-7, the SX-7 and SX-7C are comparable in the sustained performance of the GPR using eight processors.

On the SX-7 and SX-7C, when an application uses eight processors in a node, each processor can use 2K banks in the SX-7 and 0.5K banks in the SX-7C. As discussed in Section 2.1, the SX-7C does not have the margin in the number of banks. In the case of non-stride 1 memory accesses, the processing time on eight processors of the SX-7C increases due to the memory access latency not hidden by the interleaved memory. On the other hand, the SX-7 has the margin in the number of banks, and the SX-7 is superior to the SX-7C in terms of the capability to hide the memory access latency. Therefore, the SX-7C would require more banks for more effective computing and scalable performance when using entire processors of one node.

We investigate the effect of memory bandwidth per processor on the non-overlapped memory access time of the SX-7. Table 5 shows the results of relative memory access times on each application when the memory bandwidth of the SX-7 is reduced by partially shutting off network switches between processors and memory units. When the memory bandwidth is adjusted to 1/2 or 2

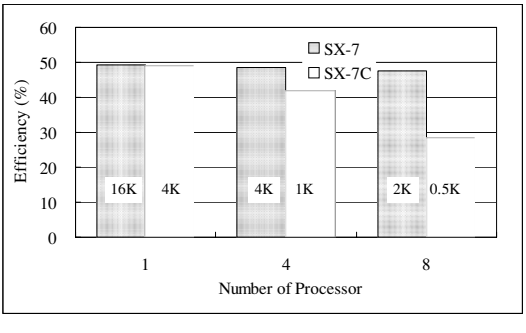


Fig. 6. Efficiency of GPR simulation on SX-7 and SX-7C: The number in each bar indicates the number of banks per processor

Table 5. Relative memory access time of five simulation codes normalized by the 4 B/FLOP case on the SX-7

B/FLOP	GPR	APFA	PRF	SFHT	PBM
4	1.0	1.0	1.0	1.0	1.0
2	3.5	3.0	2.2	3.5	75.6
1	9.5	11.5	5.7	10.1	316.7

B/FLOP, the memory access time is two or more times longer than that of the 4 B/FLOP case. When the memory bandwidth is reduced to 1/4 or 1 B/FLOP, the memory access time is four or more times longer than that of the 4 B/FLOP, which is almost comparable to the cases of the scalar platforms. As the memory bandwidth decreases, the memory read/write time increases, and the memory access time is not hidden by the pipelined vector operations. In the PBM simulation, the memory access time not hidden by vector operations is 316 times as long as that of the 4 B/FLOP case. Therefore, the sustained performance is seriously affected by the B/FLOP rates, and a memory bandwidth of the 4 B/FLOP is essential to keep the superiority of the vector platforms against the scalar platforms.

4.4 Discussion on the Memory Performance of TX7/i9510 and Altix3700

The performance of the TX7/i9510 and Altix3700 depends on the cache hit rate. Figure 7 presents the correlation between the cache hit rate and the ratio of the memory access time to processing time of the five applications on one processor; here, the cache hit rate is a sum of the L2 and L3 caches. The ratio of non-hidden memory access time becomes more than 50% even when the cache hit rate is 95%. Therefore, the cache hit rate needs to be almost 100% to achieve the high computational efficiency on the cache based platforms.

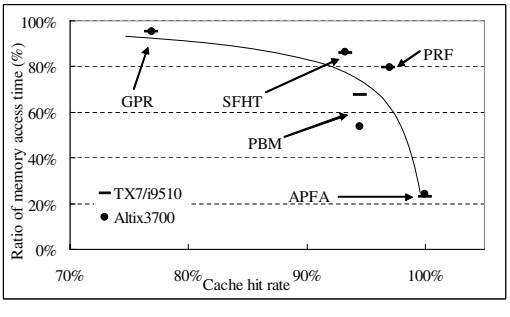


Fig. 7. Correlation of cache hit rate, ratio of memory access time to processing time on the TX7 and Altix

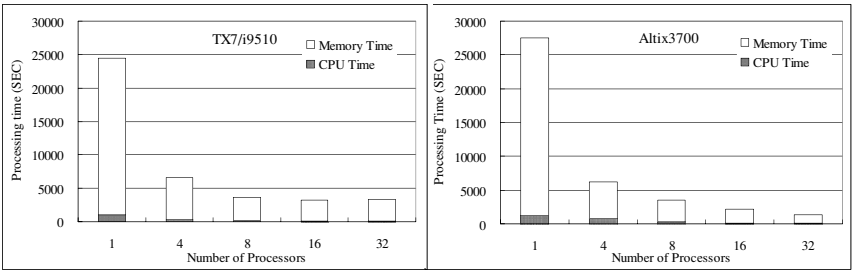


Fig. 8. GPR simulation processing time of TX7 and Altix

We measure the memory access times of GPR simulations on the TX7 and Altix. The GPR simulation has a low cache hit rate and the memory-intensive. Figure 8 is the results of the GPR simulation, and shows that the memory access time of the TX7 does not decrease in the case of eight or more processors. On the other hand, the memory access time of the Altix decreases constantly. This is because the system buses of the TX-7 connecting processors to memory are saturated with the data transfers. On the TX7 and Altix, a cell card contains processors and a main memory, which are interconnected by a 6.4GB/s bus. The cell cards of the TX7 and Altix contain four processors and two processors, respectively. In the experiment, the TX7 and Altix consist of eight cell cards and 32 cell cards, respectively. In the case of eight or more processors, the TX7 uses two or more processors per cell, and the bus of the TX7 is more likely to saturate with data transfers between processors and a memory than that of the Altix, because two or more processors of a TX7 cell share the 6.4GB/s bus. Therefore, the experimental results suggest that it is necessary for system configurations of the cache based platforms not to saturate the bus with data transfers on the bus, and the cell card of the TX7 should have two processors at a maximum.

Similarly, we measure the non-hidden memory access times of APFA simulations on the TX7 and Altix. The APFA simulation has a high cache hit rate, and the ratio of operations to memory references is 2.2. Thus, the APFA is less

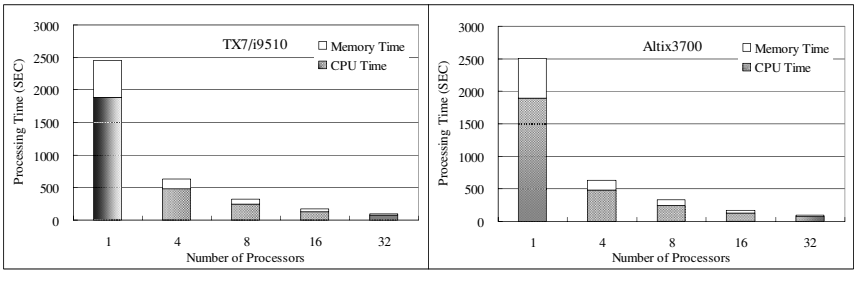


Fig. 9. APFA simulation processing time of TX7 and Altix

memory-intensive than the GPR. The experimental results of the APFA shown in Figure 9 indicate that the non-hidden memory access time of the TX7 and Altix decreases in the case of eight or more processors. In this case, the buses of the TX7 are not saturated with data transfers between processors and a memory.

5 Summary

This paper has presented the memory performance of the vector platforms of the SX-7 and SX-7C and compared it against the cache based scalar platforms of the TX7/i9510 and Altix3700. We have examined five science applications from three areas. The experimental results show that the vector platforms achieve the high efficiency and significantly outperformed the scalar platforms. We have quantitatively presented that the most important factor affecting the computational performance on science applications is the memory performance. The vector platforms use the interleaved memory systems, and their memory access latencies can be hidden by pipelined vector operations. We have confirmed that the high performance of the vector platforms is obtained due to a high memory bandwidth and a large number of banks. Our experiments using practical application codes have shown that both a balanced performance of 4 B/FLOP and the enough number of memory banks that exceeds the minimum number of banks to hide the bank cycle time are essential to achieve the higher sustained performance.

On the scalar platforms, the computational performance depends mainly on cache hit rates. We have quantitatively presented the correlation between the cache hit rate and the ratio of the memory access time to processing time of the five applications, and have confirmed that the cache hit rate must be almost 100% to achieve efficient computing. Additionally, the computational performance also depends on the performance of the memory bus that connects processors and memory in a cell card. We have demonstrated that the buses of the TX7 are saturated with data transfers among processors, when two or more processors share a 6.4GB/s bus, whereas the exclusive use of the bus by a single processor leads to the scalable performance. To avoid such a situation the scalar platforms would require maintaining the bus bandwidth per processor not to saturate the bus with data transfers and using cache effectively.

In this paper, the original codes of the five applications were used for the evaluation of the scalar and vectors platforms. In the future work, we will evaluate the performance when optimizing the codes for the scalar platforms.

Acknowledgments. The authors would like to gratefully thank: Dr. M. Sato, Dr. A. Hasegawa, Professor G. Masuya, Professor T. Ota, Professor K. Sawaya of Tohoku University, for their advice about applications. We also would like to thank Yoshiaki Matsumura and Manabu Ito for their assistance in experiments.

References

1. S. Shingu et al, A 26.58 Tflops Global Atmospheric Simulation with the Spectral Transform Method on the Earth Simulator, Proceedings of the ACM/IEEE SC2002 conference, 2002.
2. M. Yokokawa et al, 16.4-Tflops Direct Numerical Simulation of Turbulence by a Fourier Spectral Method on the Earth, Proceedings of the ACM/IEEE SC2002 conference, 2002.
3. L. Oliker et al, Evaluation of Cache-based Superscalar and Cacheless Vector Architectures for Scientific Computations, Proceedings of the ACM/IEEE SC2003 conference, 2003.
4. L. Oliker et al, Scientific Computations on Modern Parallel Vector System, Proceedings of the ACM/IEEE SC2004 conference, 2004.
5. R. A. Fatoohi, Vector Performance Analysis of Three Supercomputers: Cray-2, Cray Y-MP, and ETA10-Q, Proceedings of Supercomputing '89, 1989.
6. R. A. Fatoohi, Vector Performance Analysis of The NEC SX-2, Proceedings of Supercomputing '90, 1990.
7. H. Shan et al, Performance Characteristics of the Cray X1 and Their Implications for Application Performance Tuning, Proceedings of the ICS2004, 2004
8. K. Kitagawa et al, A Hardware Overview of SX-6 and SX-7 Supercomputer, NEC Research & Development, 44, 2-7, 2003
9. T. Senta et al, Itanium2 32-way Server System Architecture, NEC Research & Development, 44, 8-12, 2003
10. T. Kobayashi et al, FDTD simulation on array antenna SAR-GPR for land mine detection, Proceeding of SSR2003: 1st International Symposium on Systems and Human Science, Osaka, Japan, 279-283, November, 2003
11. K. S. Kunz and R. J. Luebbers, The Finite Difference Time Domain Method for Electromagnetics, Boca Raton, FL, CRC Press, 1993
12. Y. Takagi et al, Study of High Gain and Broadband Antipodal Fermi Anenna with Corrugation, 2004 International Symposium on Antennas and Propagation, vol. 1, 69-72, 2004
13. K. Tsuboi and G. Masuya, Direct Numerical Simulations for Instabilities of Remixed Planar Flames, The Fourth Asia-Pacific Conference on Combustion, Nanjing, China, November, 2003
14. M. Nakajima et al, Numerical Simulation of Three-Dimensional Separated Flow and Heat Transfer around Staggerd Surface-Mounted Rectangular Blocks in a Channel, Numerical Heat Transfer, Part A, 47, 691-708, 2005
15. K. Ariyoshi et al, Spatial variation in propagation speed of postseismic slip on the subducting plate boundary, 2nd Water Dynamics, B-30, Sendai, Japan, 2004

Optimisation of the Parallel Performance of a 3D Device Simulator for HEMTs

N. Seoane and A.J. García-Loureiro

Department of Electronics and Computer Engineering
Univ. Santiago de Compostela, 15782 Campus Sur, Spain
{natalia, antonio}@dec.usc.es

Abstract. The resolution of the linear systems generated by the discretisation of partial differential equations in semiconductor device simulation is the most time consuming part of the simulation process. In this paper we have presented an optimisation proposal of the linear systems resolution procedure used in the PSPARSLIB library. The linear systems employed in this work arise from a three–dimensional parallel simulator of semiconductor devices, specifically HEMTs, based on the drift–diffusion model. This optimisation increases the parallel efficiency of the simulation process and improves its scalability.

1 Introduction

Three–dimensional numerical simulation of semiconductor devices is extremely demanding in term of computational time because it involves complex numerical schemes. The large amount of memory and floating point operations needed make necessary the use of parallel machines and appropriate algorithms in order to obtain the maximum performance and reduce simulation times.

In this work, we have used a 3D parallel device simulator [1] for HEMTs (High Electron Mobility Transistors) [2], based on the drift-diffusion (D–D) approach to the semiconductor transport. This approach constitutes a system of coupled, nonlinear partial differential equations that have been discretised using finite element methods. Domain decomposition methods, implemented by the PSPARSLIB library [3], have been used to solve the linear systems arising from the linearisation of these equations.

In this paper, an optimisation proposal of the linear system resolution stage implemented with the PSPARSLIB library is presented. For this purpose, an analysis of the most time consuming parts of this stage, which is the main contribution to the simulation time, has been carried out in order to find the parameters with the most severe influence in its parallel efficiency, being the main goal the minimisation of the execution time of the 3D simulator.

This paper is organised as follows. Section 2 presents the mathematical expressions of the D–D transport model and briefly summarises the simulation process. Section 3 reviews the linear system solving methods used in this paper, it describes the main characteristics of the PSPARSLIB library and presents a modification proposal to optimise the resolution of the linear systems. Results obtained are presented in Section 4 while conclusions are drawn up in Section 5.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 859–868, 2006.
© Springer-Verlag Berlin Heidelberg 2006

2 Mathematical Model

In this work we have used a 3D device simulator based on the D–D approach to the semiconductor transport. The mathematical expressions of this model, a brief description of the simulation process and its parallel implementation are shown below. The equations of the drift–diffusion model are the Poisson equation and the continuity equations for electrons and holes. These coupled and nonlinear equations describe the relation between the electrostatic potential and the densities of the charge carriers in a semiconductor device. In stationary state they can be written in the following form:

$$\begin{cases} -div(\epsilon\nabla\phi) + q[n(\phi,\phi_n) - p(\phi,\phi_p) - N_D^+ + N_A^-] = 0 \\ -div(q\mu_n n(\phi,\phi_n)\nabla\phi_n) + qGR(\phi,\phi_n,\phi_p) = 0 \\ -div(q\mu_p p(\phi,\phi_p)\nabla\phi_p) - qGR(\phi,\phi_n,\phi_p) = 0 \end{cases} \quad (1)$$

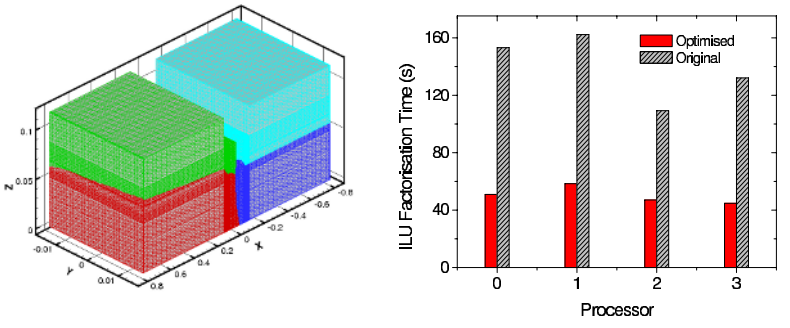
The unknowns of the problem are ϕ, the electrostatic potential, ϕ_n, the quasi–Fermi level for the electrons and ϕ_p, the quasi–Fermi level for the holes.

In the simulation process, drift–diffusion equations are discretised using the finite element method (FEM) [4] on an unstructured tetrahedral mesh. The partition of the mesh into subdomains is performed using the program METIS [5]. The same program was used to relabel the nodes in subdomains in order to obtain a more suitabl erearrangement to reduce the fill–in of the matrix. The solution scheme to solve the system of non–linear equations (1) consists of decoupling the equations using the Gummel method and linearising them using Newton's algorithm. All these resolution techniques are implemented fully in a parallel manner. The linear system associated with the Poisson equation is in general well–conditioned. However, the linear systems associated with the continuity equations cause significant difficulties because the matrices associated with these equations have high condition numbers and are badly conditioned [6].

3 Solution of Linear Systems of Equations

The part of the simulation process that requires more computational time is the resolution of the linear systems associated with the drift–diffusion model. In general, linear systems of equations can be expressed as $Ax = b$, where A in our study is a sparse and non–symmetric matrix. The basic strategies used to solve sparse linear systems are based on direct or iterative methods. Most direct methods for sparse linear systems perform a LU factorisation of the original matrix and try to reduce cost by minimising the fill–in, that is the nonzero elements introduced during the elimination process in positions which were initially zeros.

Krylov subspace methods are considered to be among the most important iterative techniques available for solving large linear systems. Iterative methods are usually combined with preconditioners to improve the convergence rates. Especially for ill–conditioned matrices, iterative methods fail without the application of a preconditioner. Two of the most common preconditioning techniques are ILU factorisations and domain decomposition methods.



(a) (b)

Fig. 1. 1(a) Example of a tetrahedral mesh of a HEMT device with 126,166 nodes divided in four subdomains. 1(b) Time employed by each processor to perform an ILU factorisation in both versions of the code.

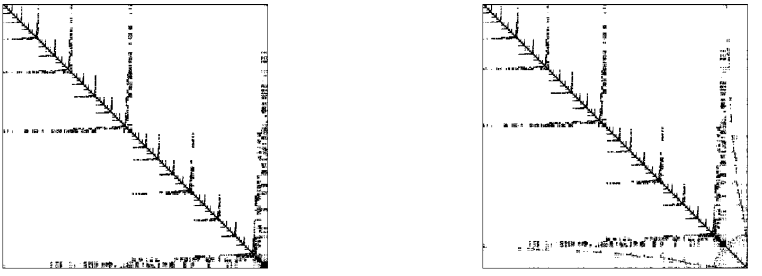
ILU factorisations are based on algebraic manipulations of the matrix to obtain some kind of approximation to the inverse matrix. The matrix **A** is factorised but without introducing all the fill–in that is produced during this process. For example, in the ILU($fill, \tau$) factorisations two criteria to introduce the fill–in are used, the position inside the matrix and a numerical threshold.

Domain decomposition methods refer to a collection of techniques which revolve around the principle of divide and conquer. If we consider the problem of solving an equation on a domain Ω partitioned into p subdomains Ω_i, the domain decomposition methods attempt to solve the problem on the entire domain by a problem solution on each subdomain Ω_i. Each node belonging to a subdomain is an unknown of the problem. It is important to distinguish between three types of unknowns: interior nodes are those that are coupled only with local nodes, local interface nodes are those coupled with external nodes as well as local nodes, and external interface nodes are those nodes in other subdomains which are coupled with local nodes. Examples of preconditioning techniques based on domain decomposition are Additive Schwarz, Multicolor SOR and Schur complement methods.

3.1 Optimisation Proposal for Solving Sparse Linear Systems

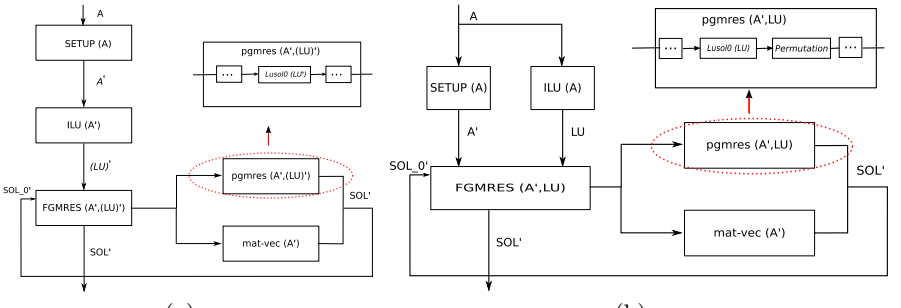
In the 3D simulator, PSPARSLIB library has been employed to solve the linear systems of equations. The linear system is firstly partitioned, then split according to the partitioning, a distributed data structure is constructed and, finally, a preconditioned Krylov solver is invoked for its solution. It uses domain decomposition preconditioners, such as Additive Schwarz, Multicolor SOR and Schur complement methods [7].

Previously, for matrices arising from semiconductor device simulation, an analysis of the performance of solution methods and preconditioning techniques employed in PSPARSLIB has been carried out [8], and the lowest execution



(a) (b)

Fig. 2. Pattern of a local matrix reordered with the METIS program previously to the call of the SETUP function (Fig. 2(a)), and the same matrix reordered with the SETUP function (Fig. 2(b))



(a) (b)

Fig. 3. Flux diagram of the initial linear systems solving stage (Fig. 3(a)) and of the optimised one (Fig. 3(b))

times were obtained with the Additive Schwarz method [9]. This algorithm is similar to a block–Jacobi iteration and consists of updating all the new components from the same residual. Assuming that A_i is the local matrix of the linear system of equations to be solved on a particular subdomain Ω_i, and x_i represents the local solution, the basic Additive Schwarz iteration works as follows:

1.- Obtain the external interface nodes $y_{i,ext}$
2.- Compute local residual $r_i = (b - Ax)_i$
3.- Solve the local linear system $A_i \Delta_i = r_i$
4.- Update the solution $x_i = x_i + \Delta_i$

In our case, a standard Incomplete LU factorisation with Threshold (ILUT) preconditioner combined with Flexible Generalised Minimal Residual method (FGMRES) is used to solve the linear system $A_i \Delta_i = r_i$ for each of the blocks.

The main goal of this work is to study how to improve the parallel efficiency of the 3D D–D simulator in order to reduce the simulation time. For this purpose, the linear systems solving stage of the simulator has been analysed to find its

Table 1. General information about the meshes employed in the simulation

Mesh	Nodes	Tetrahedrons	NNZ	Mesher
S	29,012	147,682	398,102	QMG
M	76,446	433,824	1,116,664	MMG
L	126,166	723,040	1,852,656	MMG
H	221,760	1,253,760	3,223,110	MMG

Table 2. For the S mesh, dependence with the number of processors of the average times needed for: an incomplete LU factorisation (t_{ILU}), the FGMRES solver to achieve the convergence ($t_{iter.method}$) and the solution of a local linear system (t_{solver}). It is also shown the average number of iterations of the inner solver (it_{solver}).

proc	t_{ILU}	t_{ILU-O}	$t_{iter.method}$	$t_{iter.method-O}$	t_{solver}	$t_{solver-O}$	it_{solver}
1	20.76	20.61	0.45	0.45	21.21	21.06	2
2	9.82	6.71	5.28	7.55	15.10	14.27	33
4	4.62	1.64	7.21	3.25	11.84	4.90	39
8	1.95	0.51	4.38	1.01	6.34	1.52	41
16	0.67	0.15	1.38	0.37	2.05	0.52	55
32	0.21	0.05	0.49	0.17	0.70	0.22	67
62	0.06	0.03	0.13	0.08	0.19	0.11	61

most time consuming part. For a low number of processors, the incomplete LU factorisations are one of the most important contributions to the total simulation time, limiting the improvement of the parallel performance. As an example, results filled in with slashes in Figure 1(b) (*Original*) show, for a mesh with 126,166 nodes divided in 4 subdomains, the incomplete LU factorisation time for each employed processor. The solution time of an incomplete LU factorisation is in average 150 s, being the total time of the solution of a local linear system 250 s, and between the fastest and the slowest processors there is a difference in time of roughly 48%. This unbalanced behaviour is mainly due to an internal reordering made by the PSPARSLIB library. This library needs to call a function (SETUP) before performing the ILU factorisation. This function reorders the nodes of the local matrices in such a way that it first labels the internal nodes, then the local interface ones and finally the external interface ones. This reordering changes the pattern of the local matrix, that had been optimised in arrow format with the reordering made by METIS in previous stages of the simulation, and increases its fill–in. Figures 2(a) and 2(b) represent the pattern of a local matrix before and after calling the SETUP function, respectively.

PSPARSLIB uses the SETUP function to overlap computations and communications in the following matrix–vector products since this reordering puts external interface nodes at the end of the structure sorting them by processors. Figure 3(a) shows a flux diagram of the linear systems solving stage implemented with the PSPARSLIB library. As we stated above, to solve the local nodes within each subdomain we have employed a FGMRES iterative algorithm. This method is preconditioned by a PGMRES iterative procedure, which is a simple version of the ILUT preconditioned GMRES algorithm.

In our optimisation proposal we have tried to reduce the ILU time but without compromising this overlapping of tasks. Therefore, for the ILU factorisation, the

Table 3. Same results as the ones shown in Table 2 but using the M mesh

proc	t_{ILU}	t_{ILU-O}	$t_{iter.method}$	$t_{iter.method-O}$	t_{solver}	$t_{solver-O}$	it_{solver}
2	113.20	75.90	39.11	30.47	152.32	106.37	32
4	56.04	23.40	46.89	32.09	102.93	55.49	48
8	30.54	5.27	58.92	17.13	89.46	22.40	60
16	9.35	1.23	18.87	5.11	28.22	6.34	69
32	2.63	0.35	8.86	1.25	11.49	1.60	82
62	0.65	0.11	1.84	0.47	2.49	0.58	93

Table 4. Same results as the ones represented in the two previous tables but using the L mesh

proc	t_{ILU}	t_{ILU-O}	$t_{iter.method}$	$t_{iter.method-O}$	t_{solver}	$t_{solver-O}$	it_{solver}
4	139.19	50.29	115.28	62.83	254.47	113.12	83
8	62.88	14.47	83.78	44.08	146.66	58.55	104
16	24.69	3.45	51.28	21.32	75.97	24.77	110
32	7.70	0.85	26.10	6.92	33.80	7.77	129
62	1.93	0.26	7.31	1.16	9.24	1.42	130

initial matrix (previous to the one originated with the SETUP routine) has been used, whereas in the FGMRES iterative procedure the new reordered matrix has been employed. So, after every ILU factorisation it is necessary to permutate the matrix resulting of this process in order to adapt it to the new labelling originated with the SETUP reordering. Figure 3(b) shows a flux diagram of the optimised linear systems solving stage.

This optimisation technique decreases noticeably the ILU factorisation time per processor and improves the balancing of computational effort between processors. For example, Figure 1(b) also shows the ILU time for each one of the processors employed in the optimised version. The incomplete LU factorisation times are in average 50 s and between the slowest and the fastest processors there is a difference of approximately 30%.

Therefore, the main objective of this optimisation is the use, in the resolution of an ILU factorisation, a matrix with a lower bandwidth per row than the original one. This is important due to the fact that the bandwidth per row fixes the extreme positions of a row where fill–in elements can be introduced. Hence, this optimisation permits to work with lower values of fill–in, saving computational time and memory usage.

Table 5. Influence of the number of processors in the time needed to obtain the solution of the Poisson equation in equilibrium, for the two versions of the code

	mesh S		mesh M		mesh L	
proc	t_{equi}	t_{equi-O}	t_{equi}	t_{equi-O}	t_{equi}	t_{equi-O}
1	570.53	568.68	–	–	–	–
2	460.34	377.45	4236.52	3665.68	–	–
4	339.26	132.81	3728.74	1481.93	6216.41	2930.07
8	164.02	43.49	2088.04	678.61	3654.45	1537.60
16	55.35	16.54	829.02	150.19	2148.47	534.75
32	19.47	7.71	265.75	38.88	868.13	169.43

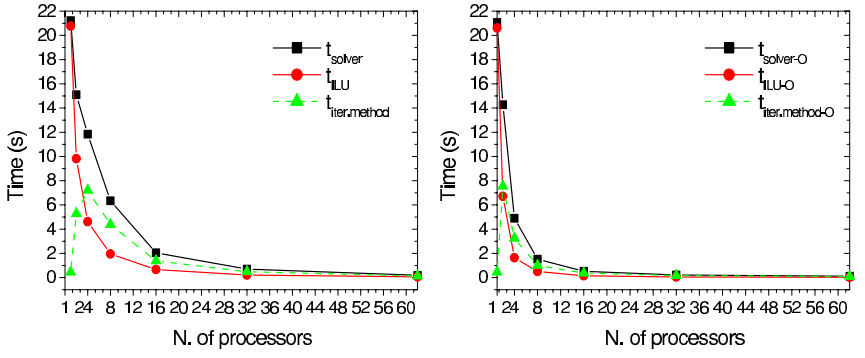


Fig. 4. These figures represent, for mesh S, t_{solver}, t_{ILU} and $t_{iter.method}$ versus number of processors, for the initial and optimised versions of the code respectively

In the optimised version, as we have seen in the flux diagram, the ILU function receives as input a matrix reordered with the METIS program (A), different from the one given by the SETUP function (A'), in order to reduce the fill–in. The only cost of this optimisation, per iteration, is the computing time necessary to permute the vector of the solution of the Lusol0 function, to adapt it to the original ordering given by PSPARSLIB. The Lusol0 function performs a forward followed by a backward triangular solve for a LU matrix, therefore this function receives as input the output of the ILU function (LU).

4 Numerical Results

The numerical results have been obtained in an HP Superdome Cluster formed by two HP Integrity Superdome servers, each with 64 Itanium2 1.5 GHz, 6 MB cache processors. The meshing in the 3D simulator is carried out using two programs, the QMG [10] and the MMG [11]. An example of a tetrahedral mesh, divided in four subdomains, arising from the MMG program is shown in Figure 1(a). To accomplish our study we have employed four meshes with different size. Their main characteristics are shown in Table 1. The results we are going to present from now on are comparatives between the two versions of the code, initial and optimised. Both versions are distinguished through the inclusion in the labelling of the optimised version results a $-O$ symbol. We have focused this study on the solution of the Poisson equation in equilibrium, although in the optimised version similar results have been found for the solution of the electron continuity equation. A fixed value of $fill = 700$ has been employed, being $2 \cdot fill$ the maximum number of fill–in elements per row that can be introduced in the structure of outgoing data.

Tables 2, 3 and 4 show for the meshes S, M and L respectively, the dependence with the number of processors employed on: the average time of performing an

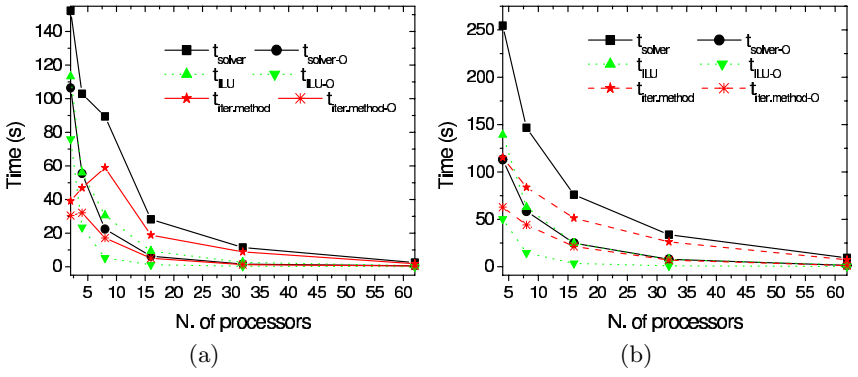


Fig. 5. Comparative, for the M and L meshes respectively, between t_{solver}, t_{ILU} and $t_{iter.method}$ versus number of processors for the two versions of the code

incomplete LU factorisation (t_{ILU}), the average time needed by the FGMRES solver to achieve the convergence ($t_{iter.method}$), the average time of solving a local linear system (t_{solver}), being $t_{solver} = t_{ILU} + t_{iter.method}$, and the average number of iterations of the inner solver (it_{solver}). This value is common for both versions of the code.

The minimun number of processors that can be employed in each one of the analysed cases depends on the size of the mesh and on the memory requirements. So, for the M mesh it is only possible to obtain results for more than 1 processor and for the L mesh it is necessary to employ at least 4 processors. As expected, in the sequential case, both versions of the code, initial and optimised, produce the same timing results since our optimisation does not change anything.

In the sequential case, t_{ILU} is the main contribution to t_{solver}, however, with the increase in the number of processors, and due to the reduction in the size of the local matrices, there is a drastical drop in the factorisation time t_{ILU}. The resolution of an incomplete LU factorisation is a task very well parallelizable in both versions of the code, although factorisation times are always lower in the optimised version of the code. On the other hand, $t_{iter.method}$ in the sequential case is almost negligible in comparison with the contribution of the ILU time. This is due to the low number of iterations that the iterative method has to carry out to achieve the convergence of the system. However, the influence of this time to t_{solver} increases with the number of processors because of the increase of the number of inner solver iterations. This increase of the number of iterations causes that $t_{iter.method}$ for 2 processors is higher than the one obtained in the sequential case, although this time scales well with an increase in the number of processors. For the S mesh, considering the optimised version, $t_{iter.method}$ becomes lower than the sequential time for more than 8 processors. This is not true in the initial version of the code, where $t_{iter.method}$ is only lower than the sequential time using 62 processors. Also, for this mesh, Figures 4(a) and 4(b) show the contributions of t_{ILU} and $t_{iter.method}$ to t_{solver}, for the initial and optimised versions of the code respectively, and their dependence with

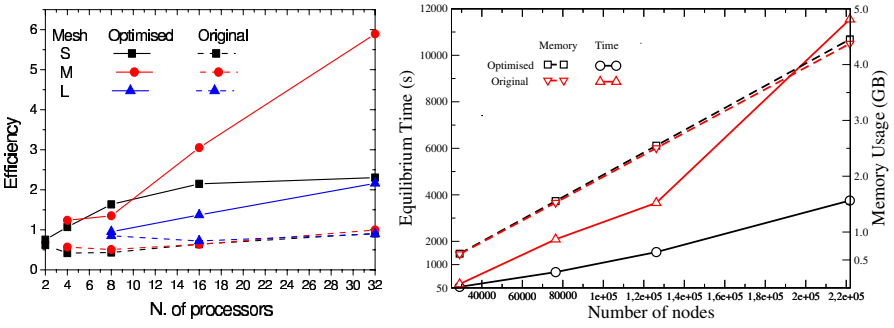


Fig. 6. For the two versions of the code, Fig. 6(a) shows the parallel efficiency, for the S, M and L meshes, obtained in the solution of the Poisson equation in equilibrium and Fig. 6(b) presents the dependence of simulation time and memory usage with the number of nodes of the mesh

the number of processors employed. Figures 5(a) and 5(b) show, for the meshes M and L respectively, a graphical comparative between t_{ILU}, $t_{iter.method}$ and t_{solver} for the two versions of the code. These figures are useful to notice the improvements, in the simulation time and therefore in the performance, given by the optimised version of the code. These improvements are not only important for the resolution of the incomplete factorisations but are also important for achieving a fast convergence in the iterative method. Table 5 shows the impact of the number of processors in the solution time for the Poisson equation for the S, M and L meshes. These times were obtained for both versions of the code, initial and optimised and do not represent average times but the total time used in the resolution of the non–linear systems needed to obtain the global solution of the problem. Numerical results emphasize the improvements in time and scalability of the optimised version in comparison to the initial one. Figure 6(a) represents the parallel efficiency for these three meshes and its dependence with the number of processors. Both versions of the code are shown in the figure for comparative purposes. The optimised version achieves super-linear efficiency up to 32 processors whereas in the initial version the increase in the efficiency is less important and only using 32 processors the efficiency reaches super–efficiency values. The parallel efficiency for the S mesh reaches a saturation value for a high number of processors. This behaviour is due to the reduction with the number of processors of the size of the local subdomains and the increase of the communications. Finally, Figure 6(b) represents for the four studied meshes and for the two versions of the code, using 8 processors, the simulation time and memory usage against the size of the problem. The utilised memory increases linearly with the number of mesh nodes, and both versions give almost identical results. However, the increase in simulation time is much more pronounced in the initial version than in the optimised one, especially for high number of mesh nodes.

5 Conclusions

In this paper we have presented an optimisation proposal of the linear system resolution stage, implemented with the PSPARSLIB library, of a 3D parallel simulator of HEMTs, although this is a general strategy and it can be useful in other scientific fields.

This optimisation increases the parallel efficiency of the simulation process and improves its scalability, leading to superlinear efficiency values. For more than 1 processor and up to 62, it reduces drastically the simulation times for the solution of the Poisson equation in equilibrium. Furthermore, similar reductions in timing results have been obtained for the solution of the electron continuity equation.

Acknowledgements. This work is partly supported by the Spanish Government under the project TIN2004-07797-C02. We are particular grateful to CESGA (Galician Supercomputing Center) for providing access to the HP Superdome.

References

1. A. García Loureiro, K. Kalna, A. Asenov: 3D Parallel Simulations of Fluctuation Effects in pHEMTs. J. Comput. Electron. **2** (2003) 369-373.
2. P. Roblin, H. Rohdin: High–speed heterostructure devices. Cambridge University Press (2002).
3. Y. Saad, Gen-Ching Lo, S. Kuznetsov: PSPARLIB users manual: A portable library of parallel sparse iterative solvers. Univ. of Minnesota (1997).
4. P. A. Markowich: The stationary semiconductor device equations , Computational Microelectronics, Springer-Verlag (1986).
5. G. Karypis, V. Kumar: METIS: A software package for partitioning unstructured graphs. University of Minnesota (1997).
6. S. Rollin, O. Schenk, A. Gupta: The effects of unsymmetric matrix permutations and scalings in semiconductor device and circuit simulation. IEEE Trans. CAD Integ. Circ. Systems, **23** (2004).
7. Y. Saad: Iterative methods for sparse linear systems. PWS Publishing Co. (1996).
8. N. Seoane, A. García Loureiro: Analysis of Parallel Numerical Libraries to Solve the 3D Electron Continuity Equation. LNCS, **3036**, (2004) 590-593 .
9. P.E. Bjorstad: Multiplicative and Additive Schwarz Methods: Convergence in the 2 domain case. SIAM, Domain Decomposition Methods, PA (1989).
10. S. A. Vavasis: QMG Ref. Manual, Comp. Science Dept., Cornell University (1996).
11. M. Aldegunde: Octree-based mesh generation for the simulation of semiconductor devices. Conference on Design of Circuits and Integrated Systems(2005).

Video Shot Extraction on Parallel Architectures

Pablo Toharia[1], Oscar D. Robles[1], José L. Bosque[1,2], and Angel Rodríguez[3]

[1] Dept. de Arquitectura y Tecnología de Computadores e Inteligencia Artificial, Universidad Rey Juan Carlos (URJC)
{pablo.toharia, oscardavid.robles, joseluis.bosque}@urjc.es
[2] Dpto. de Electrónica y Computadores, Universidad de Cantabria (UC)
[3] Dept. de Tecnología Fotónica, Universidad Politécnica de Madrid (UPM)
arodri@dtf.fi.upm.es

Abstract. One of the main objectives of Content-based Multimedia Retrieval systems is the automation of the information extraction process from raw data. When dealing with video data, the first step is to perform a temporal video segmentation in order to make a shot decomposition of the video content. From a computational point of view, this is a very high demanding task and algorithm optimization must be seeked. This paper presents a comparison between two different parallel programming paradigms: shared-memory communication and distributed memory processing using the message passing paradigm. Taking into account the software solutions, experimental results are collected over two alternative parallel architectures: a shared-memory symmetric multiprocessor and a cluster. This paper analyzes the performance achieved from the viewpoints of speed and scalability.

1 Introduction

High performance computing fits in a natural way in application areas where large volumes of data are required to be managed or processed. When dealing with multimedia databases, or even with unstructured multimedia data collections, automatic techniques for extracting relevant information from raw data must be seeked in order to efficiently access it. Content-based Multimedia Retrieval (CBMR) systems provide a very useful help to users whose aim is to introduce a query in the system and retrieve those items in the data sets which look more similar [1, 2, 3]. When large volumes of data are considered, as it is very often in the case of multimedia databases it may become necessary to look for parallel solutions in order to process, store and gain access to the available items in an efficient way [4, 5, 6, 7]. Multiprocessor shared-memory architectures are being used for these purposes since a long time ago. During decades, they were the first scientists' choice when dealing with parallel implementations [8, 9, 10]. On the other hand, last decade has settled clusters as a feasible distributed solution for applications where higher levels of scalability and lower costs are required [11, 12].

When dealing with video data, the first step is to perform a temporal video segmentation in order to isolate the minimum unit with semantic meaning: shots. This work focuses on comparing the performance achieved by two alternative parallel implementations of a video shot segmentation algorithm using two different parallel programming paradigms: shared-memory communication and distributed memory processing using

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 869–883, 2006.
© Springer-Verlag Berlin Heidelberg 2006

the message passing paradigm. Taking into account the software solutions, two different parallel architectures are considered to compare both implementations: a shared-memory symmetric multiprocessor (SMP) and a distributed system. The former is a Silicon Graphics Prisma 350 with 16 processors Itanium 2 1500 MHz. The latter is based on a Class II Beowulf cluster using a mixing of commodity hardware and software with standard technology and some specific components to improve the data transferences and I/O operations [13]. This paper analyzes the performance achieved by the implementations from the viewpoints of speed and scalability. To the author's knowledge there are not any other works comparing both parallel architectures and programming paradigms for video segmentation as well as any work with a distributed implementation of the video shot extraction problem.

The contents of this paper may be broken down into a first paragraph to describe the video segmentation algorithm used for the tests (section 2), followed by a presentation of the parallel implementations carried out (section 3), the results achieved during the tests (section 4) and the conclusions obtained (section 5).

2 Shot Extraction Algorithm Overview

2.1 Sketch of the Sequential Algorithm

Video cut detection has two main challenges: to accurate delimit the start and the end of video shots and to process the video content in a more efficient way. As stated in Section 1 it is the unavoidable first step to proceed with new data in a Content-based Video Retrieval process. Depending on the work domain, these techniques can be classified in non-compressed [14, 15] and compressed video shot segmentation [16]. This paper is focused on non-compressed video segmentation since it is an interesting testbench for primitives to be also used in a retrieval stage, as in [17].

The basic idea of video cut detection algorithms is to compute the differences between consecutive frames or groups of frames. Existing techniques differ in the way these differences are computed. Figure 1 depicts a scheme of the whole process. D_i denotes the difference between the considered frame and the previous one. In the present case, the computed D_i difference values are based on several shape and color features, although other possibilities can also be considered [18, 15]. At this point, it must be noticed that the extraction of difference values does not affect the posed parallelization strategy.

The user can select the discriminating primitive depending on the video contents. The implemented features have been Zernike invariants for the shape primitive and quantified histograms for the color feature [17]. From a computational point of view Zernike invariants are more demanding than quantified histograms, so the shape primitive will be described in the following. However, the same discussion may be applied to the color primitive.

A candidate for cut is detected when difference D_i values are higher than a dynamically computed threshold Th. The expression of Th is defined by (1):

$$Th = weight \frac{\sum_{i=j-W}^{j+W} D(i)}{2W+1} \tag{1}$$

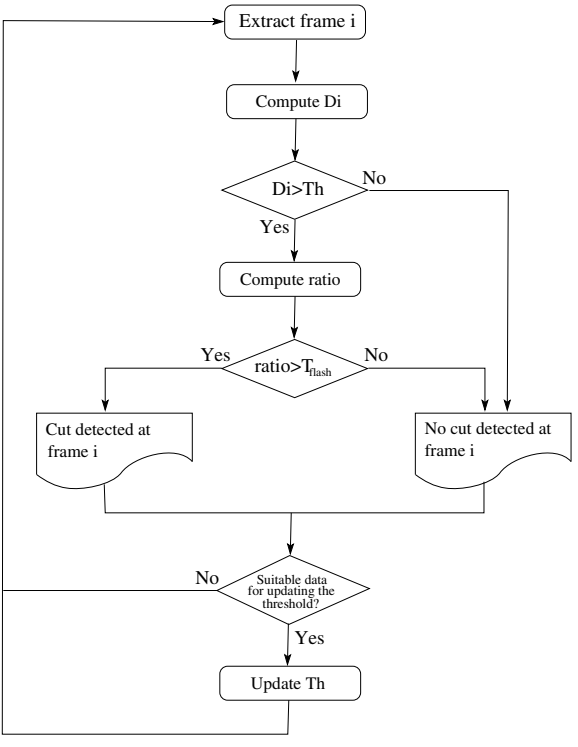


Fig. 1. Cut detection algorithm

where W is the number of difference values computed from the left and right local neighbor windows, i is the frame under consideration and *weight* is a gain factor. Therefore, threshold Th is updated for each processed frame. One of the typical artifacts present in videos is the appearance of flashes that distort normal analysis of video signals, because there is no change in the video content but abrupt changes appear in signal intensity. In order to filter out flashes, a second threshold T_{flash} has been implemented, following the model of Zhang *et al.* [19]. Finally, once comparisons are performed threshold Th is recalculated depending on the variance of the sliding window, so that if it varies too much from frame i to next frame $i+1$, as it is the case when, for example, very fast camera movements occur, the value can be adapted to the new video signal content.

Zernike invariants have been selected as shape primitive because of its demonstrated good performance in object recognition problems [20, 21]. Starting from Zernike polynomials and projecting the function over the orthogonal basis composed by the polynomials, Zernike moments can be generated as follows:

$$A_{mn} = \frac{m+1}{\pi} \sum_x \sum_y f(x,y) V^*_{nm}(x,y) dxdy \quad \text{with} \quad x^2 + y^2 \leq 1 \tag{2}$$

Table 1. Execution time and maximum speedup under Amdahl's law for different Zernike polynomials

Method	Decode time	Segmen. time	Total time	Serial frac.	Parallel frac.	Max. speedup
ZER3	3504	52452	55956	0.06	0.94	15.97
ZER5	3540	100123	103663	0.03	0.97	29.28
ZER10	3611	350609	354220	0.01	0.99	98.09

where V^*_{nm} is a set of complex polynomials defined inside a unity radius circle. From these functions, the modulus is computed to obtain the l different invariant values for each considered case. The invariant values are used to create a vector of l elements ZI_i that collect shape information from a frame i. For example, in the case of polynomials up to tenth degree, l would be 36. These vectors are used to obtain the value D_i that determines if two consecutive frames are different enough to be considered as a shot boundary:

$$D_i = \text{dist}(ZI_i, ZI_{i-1}) \tag{3}$$

where dist refers to the Euclidean distance. The expression that filters flashes is

$$\text{ratio}_{\text{flash}} = \frac{D_\text{s}}{D_i}, \tag{4}$$

where D_s is the difference between the W frames preceding the current frame and the W ones after it and D_i is the distance defined above. For the chosen Zernike moment based invariants, the expression D_s is:

$$D_\text{s} = \text{dist}\left(\frac{1}{W}\sum_{k=i-W}^{i-1} ZI_k, \frac{1}{W}\sum_{k=i+1}^{i+W} ZI_k\right) \tag{5}$$

where dist refers to the Euclidean distance. As it can be deduced from (2), invariant computation is a very high demanding task, so different order polynomials have been tested so as to verify if there were significant differences between their responses. Further details about the contents of this section can be found in [22] and [23].

2.2 Algorithm Computational Analysis

Working with video data coded in MPEG implies a first decoding stage of the compressed data and a second stage where video shot extraction is performed over the non-compressed frames following the algorithm described in Sect. 2.1.

Decoding and segmentation time values have been obtained on a 3GHz Pentium IV processor. Table 1 shows data about the execution time for a video sequence of 5000, 10000 and 20000 frames encoded at 29.97 frames per second. Several order polynomials have been tested: differences of Zernike moment invariants up to third order polynomials (ZER3), up to fifth order (ZER5) and up to tenth order (ZER10). It can be observed how the increment of the polynomial degree greatly increases the execution time of the segmentation stage.

Analyzing the execution time involved in each stage, it can be noticed that the shot boundary stage involves much larger computational load than the decoding stage even when low order polynomials are considered. Following Amdahl's law it can be computed the maximum achievable speedup when it is only considered ed a parallelization of the segmentation stage. Table 1 shows how maximum speedup is quite sensible to the polynomial degree selected. This is due to the fact that the serial fraction (i. e. the decoding time) remains constant, while the parallel fraction (i. e. the segmentation time) grows with the polynomial degree. Values in Table 1 show the benefits of using a parallel implementation, specially when using high order polynomials.

Apart from that, the decoding stage must be solved sequentially, due to the data dependencies existing among the video frames. The best way to parallelize it would be to divide the original video so that each processor would decode its own video chunk. Up to the writers knowledge, some previous works have tackled the problem of parallelizing the decoding stage for MPEG videos although these works have been based on a shared-memory platform [24, 25].

2.3 Implementation Analysis

Upon observing the operations involved in the extraction stage, it can be deduced that all processed frames have data dependency, but only those belonging to shot boundaries are critical to compute the exact points where shots start and end. This means that an approach based on data decomposition could be feasible if the boundary is fully inside the frame slice assigned to one of the p threads or processes. Therefore, the shot extraction stage could be fully parallelized.

The sequential version of the algorithm requires the displacement of a symmetric mask window with W coefficients to detect the presence of a shot boundary in the frame where the mask is placed. It implies that the assignment of frame slices to threads must be made taking into account the extra frames needed to compute mask operations in the slice's first and last frames. Due to this, there will be two small windows overlapping among consecutive slices, as Fig. 2 shows, adding $W - 1$ frames to $p - 2$ slices and $\lfloor(\frac{W}{2} + 1)\rfloor$ frames to the first and last slice.

Therefore, if we consider a video with r frames, each slice will be composed by $\lfloor(\frac{r}{p} + 1)\rfloor + \lfloor(W - 1)\rfloor$ images.

When dealing with large videos, the distributed solution based on the message passing paradigm may collapse the memory of the master processing node if there are no limitations on the amount of data sent by the master in one or successive slices. Further details about this problem will be given in the following paragraphs.

3 Parallel Implementations

As mentioned before, in this paper we deal with two different parallel architectures (SMP and cluster) and with two different parallel programming paradigms. In the following sections four different approaches are proposed. The first two are thread-based and the last two are MPI-based.

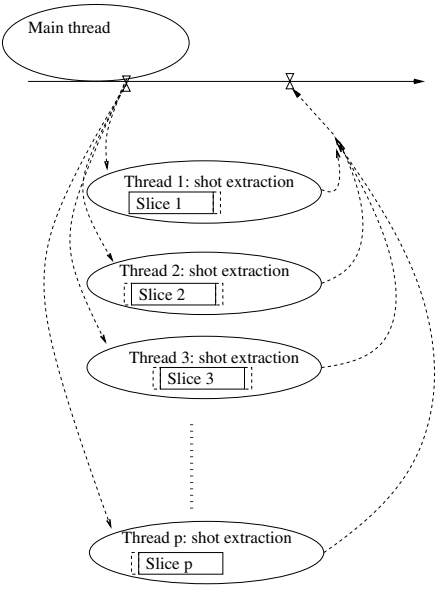


Fig. 2. Implementation strategy at thread level

3.1 Shared-Memory Paradigm

The first solution proposed is based on the shared-memory paradigm. It has been implemented using LinuxThreads [26], the available implementation of the Posix 1003.1c thread package for Linux in the Prisma SGI multiprocessor. The selection of Posix libraries to implement threads has been made considering that they are a long stable library standard, but also due to their high portability and efficiency, which guarantee good levels of software development quality. This implementation has the advantage of introducing a minimal overhead on CPU-intensive multiprocessing, reaching an assignment of almost one thread per available processor.

Two alternative approaches have been developed, differing on the data access and the decoding stage.

Distributed Decoding Approach (DDA). In this approach, each thread is in charge of data access, video decoding and shot detection tasks, so there is no need of a *master* thread. At the beginning each thread has to perform a positioning stage and once they have reached their previously assigned starting place, they can begin to compute the shot detection algorithm.

Taking into account that this algorithm is based on an adaptive threshold computed over a sliding window, each thread must begin its shot detection process some frames before the corresponding place and has to end some frames after it as well. This is the reason why assigned chunks overlap as can be seen in Fig. 2. Finally each thread generates an ordered list of detected cuts including the exact first and last frame numbers of each shot.

Centralized Decoding Approach (CDA). In this approach a *master* thread is in charge of decoding chunks of video data and passing them right after to the worker threads using shared memory. In this case the idea was to process independently both the decoding and the segmentation stages since the former has a problematic parallelization and, above all, because decoding time is much lower than segmentation time. This way, the goal is to avoid information transferences that appear in **DDA** when using data communication through the common input/output subsystem.

3.2 Message-Passing Paradigm

Implementation of parallel applications on distributed architectures, such as clusters or grids, is based on message-passing paradigm. The distributed implementation has been programmed using MPI libraries as communication primitives between *master* and *slave* processes. MPI has been selected given that it currently constitutes a *de facto* standard for message passing communications on parallel architectures, offering a good degree of portability among parallel platforms [27]. The 1.2.6-1 MPICH version developed by the Argonne National Laboratory has been chosen for the cluster implementation. It is an open-source and portable implementation of MPI [28]. This is a fully supported MPI 1.2 implementation with additional features from MPI-2 like I/O functions, cluster status and data constructors. These implementations have the advantage that can be used also on SMP architectures. In the SMP, we have employed the 7.2 LAM version from the Laboratory for Scientific Computing of Notre Dame University, a free distribution of MPI [29]. Although different MPI versions have been used in the SMP and in the cluster, independent-architecture metrics like speedup or efficiency will guarantee the validity of comparisons and analysis.

Since input data are stored in one of the cluster nodes, the most suitable solution is a farm based structure where the *master* distributes the workload among the *slave* processes and collects the partial results processed in each *slave* to obtain the video shots. In this case two alternatives are also proposed.

Static Data Distribution (SDD). The *master* process does an initial and homogeneous data distribution among the *slave* processes. Data package size is obtained dividing the total number of frames in the video by the number of available processors. Process structure in this case is very simple. The *master* begins a decoding loop, sending a complete data package to each *slave*. *Slave* processes will send the results back to the master after finishing the segmentation stage. The master gathers these partial results and stores them.

Dynamic Data Distribution (DDD). As previously stated, package size is not limited in **SDD Approach**. Thus, when dealing with large video sequences and a few *slaves* the amount of memory needed to perform the video decoding stage is very demanding. **DDD Approach** tries to solve this problem by processing fixed size packages. This way, the *master* decodes chunks of fixed size and sends each chunk to the corresponding *slave*. When the *slave* finishes processing the assigned chunk, the *master* resends another decoded chunk.

In this case, package chunks will have the same size and they will be fixed by the user. Usually, the number of packages will be greater than the number of *slave* processes.

Algorithm 1. *Master* pseudo-code

```
repeat
    Decode one package
    Send the decoded package to the slave
until there are available slaves for processing chunks of video
loop
    Wait for partial results sent from slaves
    When partial results are received, check
    if there are pending packages then
        Send a new package to the transmitter slave
    else
        Send the end label
    end if
end loop
```

Algorithm 2. *Slave* pseudo-code

```
loop
    Receive a package from the master
    Process the package
    Return the package to the master and wait for a new message from the master with a new
    package or with the end label
end loop
```

Therefore, each *slave* will process more than one data chunk. The pseudo-code of the corresponding implementation of *master* and *slave* processes is shown in Alg. 1 and 2.

Several advantages of this implementation can be stand out:

- Reduces the memory problems that appear in the previous implementation (**SDD Approach**).
- Favours dynamic load balancing among the processing nodes.
- Minimizes slaves waiting time.

On the other hand, this solution results in a more complex implementation.

4 Experimental Results

4.1 Parallel Architectures

The parallel implementations have been tested on a SMP and on a cluster described next.

Nemea: Shared-Memory Symmetric Multiprocessor. The experiments performed on the shared-memory symmetric multiprocessor have been tested on a SGI Prisma 350 machine, an SMP scalable up to 128 processors with the following configuration: 16 Intel Itanium 2 IA-64 1500 MHz microprocessors model 2 rev. 1; 32 GB DDRAM NUMAlink main memory; 4 MB L3 cache; 800 GB from 5 Serial ATA SGI TP9300 hard disks and one 70 GB ATA disk; several external storage interfaces (2 QLogic

QLA2312 Fibre Channel Adapters and one IDE adapter); network interface model SGI IO9/IO10 Gigabit Ethernet (Copper). This is a symmetric multiprocessor in which all processors access the main memory using a common address map. Access is gained through a high-speed bus with a peak bandwidth of 12.8 GB/s. The operating system is SuSe Linux 9 Enterprise with 2.6.5 kernel, using XFS journaled filesystem.

Magerit: Beowulf Cluster. The cluster setup is a subset of the available resources in the CeSVIMa center [30]. The CeSVIMa cluster is made up of 180 eServer BladeCenter JS20 nodes (168 process nodes and 12 I/O nodes) linked together using a Myrinet network with 1Gb of bandwidth connected by one e1350 Myrinet switch and one Cisco 6509 switch with 180 ports. Each of the nodes features the following configuration: 2 IBM Power 970 2.2 GHz processors and 4 GB of main memory. The cluster has additional specific I/O resources, like 2 servers for storage management model pSeries 615, with 2 Pentium IV 1.45 GHz and 8 GB of main memory, and 1 DS4100 disk controller managing two EXP100 extensions with 14 50GB SATA disks each one. The cluster operating system is SuSe Linux 9 Enterprise with 2.6.5 kernel. Jobs in the cluster are launched using IBM LoadLeveler for Linux.

4.2 Experimental Setup

Tests have consisted on running the optimized video segmentation implementations using as input different length videos and using different number of threads and cluster nodes. Video sizes tested have been 5000, 10000 and 20000 frames. As frame rate is 29.97Hz, the length of test videos is approximately 3, 5 and 11 minutes respectively. On the other hand, tests using different number of threads and cluster nodes have been run in order to measure both the speedup and the efficiency achieved on different scenarios.

In **Nemea**, several setups with 1, 2, 4, 8 and 16 processors have been tested. In **DDA Approach**, all processors run the same code. In **CDA Approach**, one of the processing nodes decodes the video sequence; when it has one package ready, it is sent to the *slave* that performs the segmentation. In the general case, each *slave* process is assigned to one of the available processors, plus one processor dedicated to run *master* code. The only exception is the setup with the maximum number of processing nodes, 16, running the *master* in one of them and 15 *slaves* processes in the rest of them.

For the message passing paradigm, we have tested the **SDD Approach** in **Nemea** and the **DDD Approach** in **Magerit**. **Nemea** implementation is based on the **SDD Approach** since it is very similar to the **CDA Approach** and this way we can compare the LinuxThread and the MPI implementations in the SMP machine. **Magerit** implementation is based on the **DDD Approach** because this version avoids the memory bottleneck that appears in the **SDD Approach**. In **Magerit**, implemented versions with 1, 2, 4, 8, 16 and 32 processors have been tested.

The planned tests allow to compare both architectures and both programming paradigms.

4.3 Results Analysis

This section collects the obtained results and the analysis done from the experimental results. The main goals of the experiments are the following:

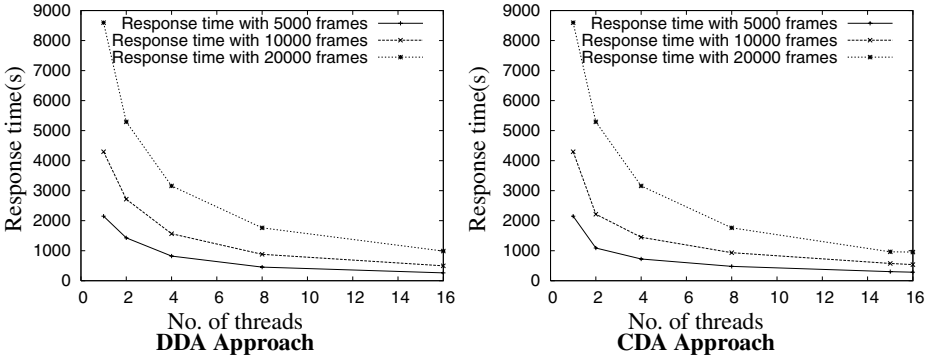


Fig. 3. Parallel application response time with LinuxThreads on **Nemea**

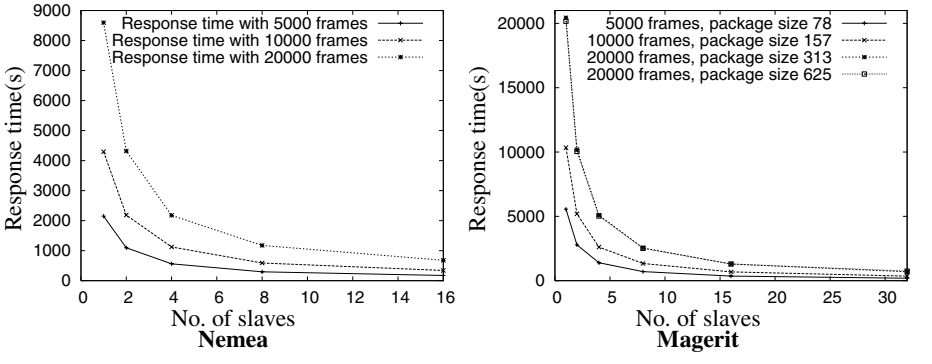


Fig. 4. Parallel application response time with MPI on **Nemea** and **Magerit** (**SDD Approach**)

- To validate the viability of a parallel solution for the video segmentation application in different architectures and with several parallel programming paradigms (shared-memory and message passing).
- To compare the performance of two alternative architectures, based on shared-memory and on distributed memory, to evaluate which one offers the best figures in this application. The comparison will take into account the response time and the communication overhead appearing in each case.
- To compare two parallel computation paradigms, like shared-memory *versus* message passing programming. In this case, it must be noticed the great influence of the selected libraries on the implementation results (LinuxThreads and MPI).

Figures 3 and 4 present the evolution of the response time for both implementations when the number of processors is increased and considering several data sizes: video sequences of 5000, 10000 and 20000 frames. The greatest packet size is fixed by the test with the maximum number of nodes. The other values are divisors of this value. Response time is defined in the classical sense as the time spent by the application from the user's point of view. All figures show a great reduction of response times, so it can be

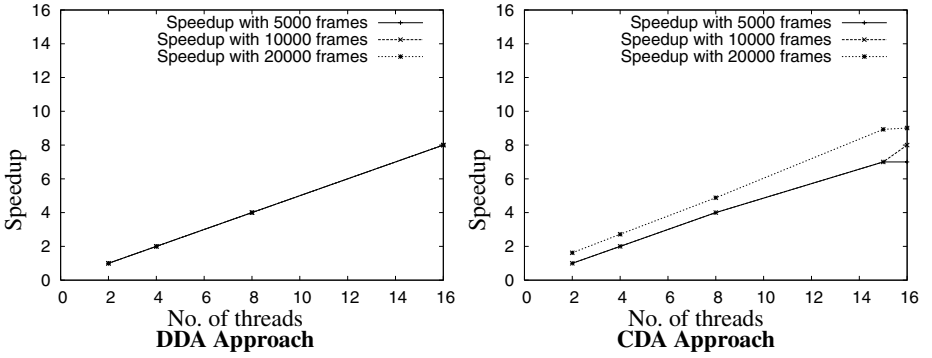


Fig. 5. Speedup of the LinuxThreads versions on **Nemea**

deduced that parallel implementations are a good solution to cut down the application response time.

A more detailed analysis of the results achieved by each implementation shows that the response time of both thread versions are very similar, as can be seen in Fig. 3. On the other hand, Fig. 4 shows very similar curves for the MPI implementation on both parallel architectures, Nemea and Magerit. It can be noticed how response time values in the SMP are quite lower than response time values obtained in the cluster when dealing with a setup with few processors: slightly under 50%. These figures can be due to the different performance achieved by the microprocessors installed in both parallel architectures. A deeper analysis of this topic is out of the scope of this work, but the reader may consult some of the available microprocessors benchmarks to look for a better justification of these values [31]. When the number of slaves in the cluster is increased, the reduction slope of the curve is sharper in **Magerit**. Better response times in **Magerit** can be achieved since the number of available processors is greater than in **Nemea**.

MPI version over the SMP architecture clearly improves LinuxThreads version on the same architecture. This improvement is into the interval [30%,50%], increasing with the number of processors in the setup and remains almost constant when data size changes. This means that the LinuxThreads synchronization mechanisms are not so optimized as the communication and synchronization primitives available in MPI. Therefore, it can be concluded that even in a shared-memory architecture, the use of MPI surpasses the performance achieved by the LinuxThread implementation.

Figures 5 and 6 present speedup values computed from the execution time of the previous figures. It can be noticed how the speedup follows a nearly linear curve as a function of the number of considered processors. Figure 5 shows a linear curve with a slope value of 0.55, although experimental results are far away from the theoretical optimum value. It must be also emphasized that speedup is almost independent of the input data size increment, therefore communication overhead is negligible with respect to processing time. Speedup curves of the MPI versions, as shown in Fig. 6, are also almost linear, but in this case with experimental values very close to the optimum ones. The slope of the speedup curve is 0.82 for the SMP and 0.91 for the cluster. Comparing both implementations, it can be asserted that the MPI implementation behaves much better than the LinuxThreads implementation in terms of speedup (Fig. 5-**CDA**

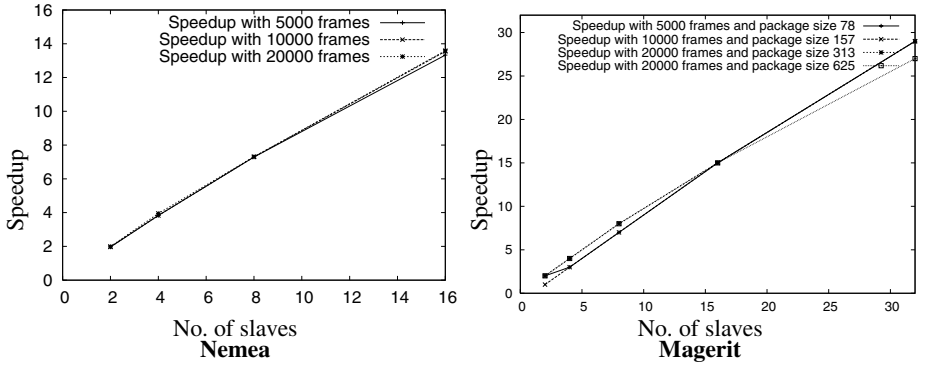


Fig. 6. Speedup with MPI on **Nemea** and **Magerit** (**SDD Approach**)

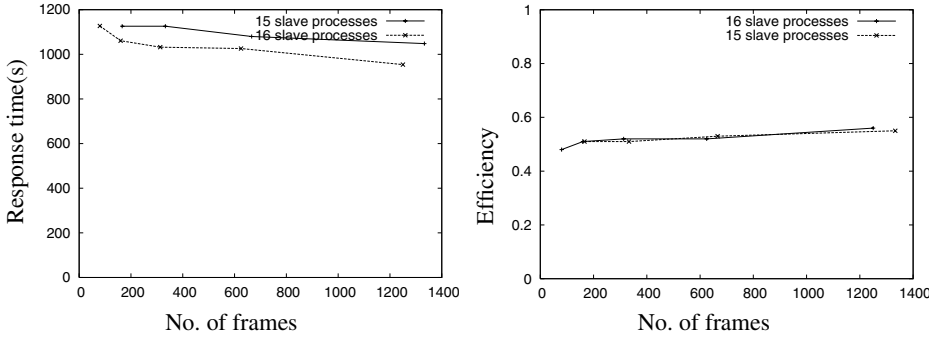


Fig. 7. Detailed analysis of the MPI version with fixed package size (**DDD Approach**)

Approah vs. Fig. 6-**Nemea**). Comparing both parallel architectures with the MPI version, the cluster is slightly more scalable than the SMP. (Fig. 6-**Nemea** vs. Fig. 6-**Magerit**).

Figure 7 shows response time and efficiency[1] values obtained when package size used in the experiments performed with the **DDD Approach** executed in **Nemea** changes. With 16 *slaves*, one of them shares the corresponding CPU with the *master*.

Analyzing the performance figures with respect the change in the data package size, measured in number of frames, it can be noticed a light but insignificant improvement. The small improvement is the result of combining two opposite effects:

- On one hand, increasing the package size reduces the global communication overhead because a fewer number must be transmitted, improving the response time.
- On the other hand, increasing the package size rises the decoding time per package, causing higher waiting time values in the *slaves*.

The net result of combining both factors produces a modest improvement of the performance taking into consideration the package size.

[1] $\text{Efficiency} = \dfrac{\text{Speedup}}{\text{Number of processors}}$.

5 Conclusions and Future Work

This paper has presented a comparison among different parallel architectures and parallel programming paradigms of a video shot segmentation algorithm used in a Content-based Multimedia Retrieval System. Programmed implementations attempt to cover all possible parallel programming aspects, just as the different studied paradigms: two LinuxThreads implementations and two MPI versions, tested on a shared memory symmetric multiprocessor and on a cluster.

The aim of the experiments is to evaluate the performance achieved by the different implementations. The main conclusions extracted from the experiments are the following:

- Shared memory architectures obtain better results with a small number of processors, but are less scalable than clusters, even considering applications demanding a very high network bandwidth, like the one tackled in this paper. This is deduced from the **Nemea** figures when dealing with two or four processors, since in these cases response times are better, but the speedup values are worse, meaning a poor system scalability.
- Beowulf clusters with very powerful networks, like Myrinet in this case, achieve excellent scalability results.
- Experiments based on message passing paradigm have obtained very good performance results, improving thread-based implementations even when they are run in the shared memory architecture.

Future work will include a deeper scalability analysis of the cluster architecture. Load balancing will be considered in cluster implementations to improve the overall performance of the video shot detection system. Further analysis of the I/O bottleneck in the shared memory architecture will be also studied to solve the existing bottleneck in the video positioning stage. Disk stripping and Redundant Arrays of Inexpensive Disks (RAID) will be considered in this study.

Acknowledgments

This work has been partially funded by the Spanish Ministry of Education and Science (grant TIC2003-08933-C02) and Government of the Community of Madrid (grants GR/SAL/0940/2004 and S-0505/DPI/0235). Data used in this work was provided by the TRECVID project, sponsored by the National Institute of Standards and Technology (NIST) with additional support from other U.S. government agencies.

References

1. del Bimbo, A.: Visual Information Retrieval. Morgan Kaufmann Publishers, (1999)
2. Marques, O., Furht, B.: Content-based Image and Video Retrieval. Kluwer Academic (2002)
3. Petkovic, M., Jonker, W.: Content-Based Video Retrieval. Springer (2003)
4. Shih, T.K., ed.: Distributed multimedia databases: techniques & applications. Idea Group Publishing (2002)

5. Srakaew, S., Alexandridis, N., Nga, P.P., Blankenship, G.: Content-based multimedia data retrieval on cluster system environment. In Sloot et al., eds.: 7^{th} International Conference HPCN Europe 1999, Springer Verlag (1999) 1235–1241
6. Kao, O., Steinert, G., Drews, F.: Scheduling aspects for image retrieval in cluster-based image databases. In Buyya et al., eds.: Proceedings First IEEE/ACM International Symposium on Cluster Computing and the Grid, IEEE Comp. Soc. (2001) 329–336
7. Bosque, J.L., Robles, O.D., Pastor, L., Rodríguez, A.: Performance analysis of a CBIR system on shared-memory systems and heterogeneous clusters. In: Proceedings on IEEE CAMP 2005, IEEE (2005) 309–314
8. Krishnamurthy, E.: Parallel Processing: Principles and Practice. Addison-Wesley (1989)
9. Hwang, K.: Advanced Computer Architecture: Parallelism, Scalability, Programmability. McGraw-Hill (1993)
10. Pitas, I., ed.: Parallel Algorithms for Digital Image Processing, Computer Vision and Neural Networks. John Wiley & Sons (1993)
11. Nupairoj, N., Ni, L.M.: Performance evaluation of some MPI implementations on workstations clusters. In: Proceedings of the SPLC94. (1994) 98–105
12. Bruck, J., Dolev, D., Ho, C.T., Rosu, M., Strong, R.: Efficient message passing interface (MPI) for parallel computing on clusters of workstations. Journal of Parallel and Distributed Computing (40) (1997) 19–34
13. The beowulf cluster site. Web (2006) Retrieved september 15, 2006, from source. www.beowulf.org
14. Porter, S., Mirmehdi, M., Thomas, B.: Temporal video segmentation and classification of edit effects. Image and Vision Computing **21**(13–14) (2003) 1097–1106
15. Valencia, G., Rodríguez, J.A., Urdiales, C., Sandoval, F.: Color-based video segmentation using interlinked irregular pyramids. Pattern Recognition **37**(2) (2004) 377–380
16. Antani, S., Kasturi, R., Jain, R.: A survey on the use of pattern recognition methods for abstraction, indexing and retrieval of images and video. Pattern Recognition **35**(4) (2002) 945–965
17. Robles, O.D., Toharia, P., Rodríguez, A., Pastor, L.: Towards a content-based video retrieval system using wavelet-based signatures. In Hamza, M.H., ed.: 7th IASTED International Conference on CGIM 2004, IASTED, ACTA Press (2004) 344–349
18. Černeková, Z., Nikou, C., Pitas, I.: Shot detection in video sequences using entropy-based metrics. In: Proc. of the IST'01, Iran (2001) 156–165 Supported by the MOUMIR project.
19. Zhang, D., Qi, W., Zhang, H.J.: A new shot boundary detection algorithm. In Shum et al., eds.: IEEE Pacific Rim Conference on Multimedia. Vol. 2195., IEEE, Springer (2001) 63–70
20. Khotanzad, A., Hong, Y.H.: Invariant image recognition by zernike moments. IEEE Transactions on Pattern Analysis and Machine Intelligence **12**(5) (1990) 489–497
21. Kamila, N.K., Mahapatra, S., Nanda, S.: Invariance image analysis using modified Zernike moments. Pattern Recognition Letters **26**(6) (2005) 747–753
22. Toharia, P., Robles, O.D., Ángel Rodríguez, Pastor, L.: Combining shape and color for automatic video cut detection. In: Proc. of the TRECVID 2005 Workshop, 336–345
23. Toharia, P., Robles, O.D., Rodríguez, A., Pastor, L.: Xml specification for avi files in a content-based video retrieval system. In Villanueva, J.J., ed.: Proceedings of the Fourth IASTED International Conference on VIIP, IASTED, ACTA Press (2004) 374–378
24. Bilas, A., Fritts, J., Singh, J.P.: Real-time parallel mpeg-2 decoding in software. In: IPPS '97, IEEE Computer Society (1997) 197–203
25. Bhandarkar, S.M., Chandrasekaran, S.R.: Parallel parsing of mpeg video on a shared-memory symmetric multiprocessor. Parallel Computing **30**(11) (2004) 1233–1276
26. Leroy, X.: The linuxthreads library. Web (2006) http://pauillac.inria.fr/ xleroy/linuxthreads/
27. MPI Forum: A Message-Passing Interface standard. Web (2006) Retrieved september 15, 2006, from source. www.mpi-forum.org

28. Argonne National Laboratory: Mpich vs. 1.2.6.1 (2004) Web (2006) Retrieved september 15, 2006, from source. http://www.mcs.anl.gov/mpi/mpich/
29. Squyres, J.M., Meyer, K.L., McNally, M.D., Lumsdaine, A.: LAM/MPI User Guide. University of Notre Dame. (1998) LAM 6.3.
30. Universidad Politécnica de Madrid (UPM) and Centro de Investigaciones Energéticas, Medioambientales y Tecnológicas (CIEMAT): Centro de supercomputación y visualización de madrid. Web (2006) Retrieved september 15, 2006, from source. www.cesvima.upm.es
31. Standard Performance Evaluation Corporation: Spec benchmarks. Web (2006) Retrieved september 15, 2006, from source. www.spec.org

Performance of Real-Time Data Scheduling Heuristics Under Data Replacement Policies and Access Patterns in Data Grids

Atakan Doğan

Department of Electrical and Electronics Engineering,
Anadolu University, 26470 Eskişehir, Turkey
atdogan@anadolu.edu.tr

Abstract. A variety of real-time data scheduling heuristics were proposed for distributed, data intensive real-time applications running on a distributed computing system. The proposed heuristics are used to produce real-time data dissemination schedules for the applications' requests for the data stored on the machines in the system. However, how these real-time data scheduling heuristics will perform for different data replacement policies and data access patterns is a question left unanswered. Based on this motivation, in this study, the performance of the two real-time data scheduling heuristics, namely the Full Path Heuristic and the Extended Partial Path Heuristic, are evaluated under different data replacement policies and data access patterns. A detailed set of simulation studies are presented to reveal how these algorithms are affected by the changes in the data replacement policy and data access pattern as well as the other system parameters of interest.

1 Introduction

Applications with real-time requirements are currently emerging in many disciplines of science and engineering, some of which are defense, scientific experimentation, monitoring and control via wireless sensor networks, intelligent transportation systems, and automotive. A common characteristic of such applications is that data produced or stored in one component of the system need to be transferred across a network of limited resources to another component (or components) while respecting associated real-time attributes. Achieving such data transfers in a timely manner and servicing as many data transfer requests as possible in a distributed environment is a nontrivial problem known as the *real-time data dissemination, data distribution, or data scheduling problem* [1]-[3].

Recently, a class of real-time data dissemination algorithms has been proposed in [1]-[3]. In [1], three different heuristics for the *Battlefield Awareness and Data Dissemination* (BADD) are proposed. Later, better algorithms in terms of maximizing the number of satisfied requests (their deadlines are met) are introduced in [2], [3]. All of these real-time data dissemination algorithms are applicable to BADD-like environments as well as to distributed computing platforms in which a set of storage facilities are serving for the applications' real-time data transfer requests. For

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 884 – 893, 2006.
© Springer-Verlag Berlin Heidelberg 2006

example, Data Grids, which harness geographically distributed storage and computing resources, are envisaged to run a variety of scientific simulation and experiment applications which involve a large number of data-intensive jobs [4]. Furthermore, these applications require the efficient management and transfer of terabytes and petabytes of information [4]. A major problem identified for peta-scale data intensive computing is how to schedule jobs and their related data in an effort to minimize jobs' completion times and bandwidth/storage space consumed due to the data transfers [5]-[7]. In order to alleviate these problems, data are dynamically replicated on multiple storage systems guided by a replication algorithm [5]-[9]. Even though efficient data replication and caching reduce the time to transfer data from a storage unit to a machine on which a requestor job (or jobs) of the data is running, it is clear that they are not sufficient to support multiple applications producing thousands of real-time data transfer requests on a Data Grid. A solution to this challenging problem is to deploy the real-time data dissemination algorithms proposed in [1]-[3] together with the replication and caching strategies in [5]-[9] so as to realize a real-time Data Grid infrastructure.

The motivation of this study is based on the following observations. The studies in [1]-[3] evaluated the performance of the respective algorithms assuming a naïve data replacement policy and totally random data access pattern only. However, according to [5]-[9], both the data replacement policy and the data access pattern can have significant impact on the performance of the system. On the other hand, the studies in [5]-[9] assessed the value of a variety of replication/data replacement strategies under different data access patterns. But, they did not consider accessing the data under some time constraints, which could be essential for satisfying some Grid users. Thus, this preliminary study tries to elaborate on the performance of the real-time data scheduling algorithms under different data replacement policies and data access patterns.

2 Problem Formulation

A *Data Grid* is composed of a set of machines $\mathbf{M}= \{M_1, M_2, ..., M_m\}$ and links $\mathbf{L}= \{L_1, L_2, ..., L_n\}$ each of which provides a bidirectional connection between two machines. Each machine M_i has a limited storage capacity C_i and each link L_j is associated with a specific bandwidth of B_j. These machines are organized in tiers, which are due to the Large Hadron Collider (LHC) tiered computing model [10]. In this study, the Data Grid is modeled to have four tiers as in [8], [9] (there are five of them in the LHC architecture). The Tier 0 is the source where all data are produced; the Tier 1 includes a few national centers around a country; each Tier 1 has a number of related Tier 2 work groups at universities or research labs; the Tier 3, finally, consists of a large number of workstations.

The Data Grid is used for running real-time distributed applications which generate a set of requests $\mathbf{R}= \{R_1, R_2, ..., R_r\}$ for data-items $\mathbf{X}$, where $\mathbf{X}= \{X_1, X_2, ..., X_q\}$ denotes the set of q unique data-items. Each request R_k is associated with one of q data-items X_k to be transferred from a source machine to a destination machine M_k, a

deadline value D_k by which data-item X_k must be delivered to its destination M_k, and a release time A_k after which the request should be considered for scheduling. Thus, request R_k is summarized by the following tuple: $\{X_k, M_k, D_k, A_k\}$. As in [8], [9], it is assumed that all data-items $\mathbf{X}$ are initially stored in the Tier 0 and that the requests can come from only the Tier 3 machines. With respect to this setting, the Tier 1 and Tier 2 machines will act just as intermediate storage sites on which data-items can be replicated during their transfer to the Tier 3 machines. Apart from the tiered computing model adopted in this study, the studies in [1]-[3] assumed a computing system in which all data are distributed among the machines of the system and each machine can generate a real-time data transfer request for a data-item which it does not already have.

In the Grid, a centralized scheduler accepts requests from the applications running on the system and creates a data dissemination schedule for the current requests under which the respective data-items will be transferred from machine to machine until their destination machines. In [1]-[3], the scheduler adopts the following staging model in establishing real-time data dissemination schedules.

Definition 1. Staging a data-item from a source machine to a destination machine is defined as moving the data-item in its entirety from the source to some intermediate machines along the path to the destination machines. Intermediate machines should keep the data-item until it is delivered to its destination machine(s).

This model is also considered as a possible replication strategy for the Data Grids in [8], which called it as the *Fast Spread*. According to [8], the Fast Spread is the best strategy among the six different replication/caching strategies studied therein in terms of minimizing the job completion times and bandwidth usage if the Data Grid users access data in a totally random manner or if their access pattern shows a small degree of temporal locality, which is at the cost of excessive storage consumption. Furthermore, under both small degree of temporal and geographical locality, it is still the best in saving the bandwidth, and it comes second in reducing the job completion times. Based on these promising results as well as the fact that it is the underlying data staging model of the algorithms in [1]-[3], the Fast Spread has been chosen as the replication strategy of this study. Cooperating with the Fast Spread, three different data replacement policies are considered. The first one is the *Least Recently Used* (LRU) which deletes the oldest referenced data-item(s) on a machine to free up large enough storage space for a new data-item. The second one is the data replacement policy of [2] (and [3]) under which once a data-item has reached to its destination, it is immediately removed from all intermediate machines along its path. The third policy is from [1] which keeps the data-item on the intermediate machines even after it has been delivered to its destination. This policy requires that the data-item be deleted from the intermediate machines at once as soon as the respective request's deadline plus some fixed amount of time has been passed. When the LRU and the latter two policies are compared, an important difference should be emphasized. The LRU is a *dynamic* policy in the sense that a data-item can be replaced on any machine at any given time. As a result, there is no blocking among the requests. On the other hand, the latter two can be considered to be *static* since they do not allow the data

replacement on the intermediate machines until a specific point of time. For example, under the policy of [2], a data-item on an intermediate machine cannot be replaced until it is delivered to its destination. Thus, a request may block another one for some time on an intermediate machine, which is directly proportional to when the blocking request's data-item is reached to its destination.

Assuming that the Fast Spread is the replication strategy, the goal of the scheduler is defined as satisfying as many request deadlines as possible in an oversubscribed Data Grid with limited storage and bandwidth capacities under a specific data replacement policy.

3 Real-Time Data Scheduling Heuristics

In this section, two real-time data scheduling heuristics, namely the Full Path Heuristic (FPH) [1] and the Extended Partial Path Heuristics (EPP) [2] are described in some detail to keep this study as a self-contained one.

3.1 Full Path Heuristic

The Full Path Heuristic proposed in [1] is built around the Dijkstra's multiple-source shortest path algorithm. The FPH runs as follows: In each iteration, it finds a least cost request among the satisfiable requests (a satisfiable request is the one for which the Dijkstra's multiple-source shortest path algorithm can find an available shortest path from a source to its destination and its deadline can be met using this shortest path); it, then, transfers the least cost request's data-item from a source machine up to the destination along the shortest path. Specifically, the FPH with the LRU policy, which is very similar to the original FPH in [1], is composed of the following steps:

1. For a given set of requests $\mathbf{R}$, repeat steps (2)-(6) until $\mathbf{R}$ is an empty set.
2. For each request $R_k \in \mathbf{R}$, run the Dijkstra's multiple-source shortest path algorithm to determine a shortest path from a set of source machines of data-item X_k to the request's destination machine M_k.
3. Mark a request as *satisfiable* if a shortest-path from a source to its destination machine meeting the request's deadline is found; otherwise, mark the request as *unsatisfiable*.
4. For each satisfiable request, compute the value of a *cost* function, which is taken as the urgency of the request in this study.
5. Determine the most urgent request and transfer the respective data-item to the destination machine along its shortest path. During the transfer, replicate the data-item on every intermediate machine (as well as on the destination) while enforcing the LRU if enough storage space to hold the data-item is not available. Update the storage space consumption on each machine along the path.
6. Drop a request from $\mathbf{R}$ if either the request is satisfied or the request's deadline cannot be met at all.

3.2 Extended Partial Path Heuristic

The Extended Partial Path Heuristic proposed in [2] is also built around the Dijkstra's multiple-source shortest path algorithm. The EPP produces schedules as follows: In each iteration, similar to the FPH, the EPP groups the available requests as satisfiable and unsatisfiable requests. Apart from the FPH, however, the EPP calls the Partial Path Heuristic (PPH) proposed in [1] under the infinite storage capacity assumption for each machine for this grouping. Among the satisfiable requests, the most urgent request together with its *extended path*, a path with usually more than one hop is determined. Then, the data-item of the most urgent request is transferred along the computed extended path towards its destination. Specifically, the EPP with the LRU, which is a little bit different from its original in [2], consists of the following steps:

1. For a given set of requests $\mathbf{R}$, repeat steps (2)-(7) until $\mathbf{R}$ is an empty set.
2. For each request $R_k \in \mathbf{R}$, run the Dijkstra's multiple-source shortest path algorithm to determine a shortest path from a set of source machines of data-item X_k to the request's destination machine M_k.
3. Call the PPH under the assumption that infinite storage capacity is available for each machine and the request's urgency is to be its cost. As an input set of requests, let the PPH have every request for which an available shortest path meeting the corresponding deadline is found in Step 2. Mark a request as *satisfiable* if its data-item can be delivered before the deadline by the PPH; otherwise, mark the request as *unsatisfiable*.
4. For each satisfiable request, compute its urgency. Determine the most urgent request (request with the minimum urgency value) and the second most urgent request among the satisfiable requests.
5. For the most urgent request, compute the length of the extended path in terms of the number of hops. The length of the extended path is determined based on the urgency value of the second most urgent request so that the most urgent request is transferred from one machine to another until the urgency value of the second most urgent request is not negative.
6. Transfer the data-item of the most urgent request by one or more hops along its extended path towards its destination. During the transfer, replicate the data-item on every intermediate machine (as well as on the destination) while enforcing the LRU if enough storage space to hold the data-item is not available. Update the storage space consumption on each machine along the path.
7. Drop a request from $\mathbf{R}$ if either the request is satisfied or the request's deadline cannot be met at all.

There is a subtle difference in enforcing the Fast Spread policy between the FPH and EPP due to their scheduling nature. The FPH transfers a data-item from a source up to the destination without scheduling another request, i.e., there is only one data-item in transit at any given time. Under the Fast Spread, the data-item is replicated on each machine along the path and each replica can later be replaced in accordance with the LRU. But, the EPP completes the transfer of a data-item in usually more than one step based on the computed extended paths and the Fast Spread will be applied to each of these steps individually. That is, a copy of the data-item is left on each

machine along the extended path, which is usually shorter than the full path. As a result, because of the capacity constraints, a data-item which has been transferred by several hops towards its destination can be replaced by another data-item in several machines.

4 Simulations

A simulator was developed to investigate the performance of the FPH and EPP together with the Fast Spread data replication strategy, and the data replacement policies of the LRU and the ones in [1] and [2]. The simulator consists of three parts. The network component is used to create LHC-like tiered computing systems. On top of the network component, the FPH and EPP are built. The final component of the simulator is the request generator which produces real-time data transfer requests.

Network: With the start of the simulation, a LHC-like tiered computing system is created. As in [8], there are four tiers. The Tier 0, 1, 2, and 3 are assumed to include 1, 4, 16, and 79 (64 in [8]) machines, respectively, which is a total of 100 machines. These machines are interconnected by a randomly generated tree topology.

The amount of data in a Grid will be in the order of peta-bytes, which is assumed to correspond to 1,000,000 unique data-items. However, it is not feasible to simulate the transfer of such a large number of data-items on a single computer. Thus, the number of data-items is scaled down from 1,000,000 to 100 using a scale of 1:10,000 as in [8]. Accordingly, the storage capacity of each machine should be scaled down as well, since the performance of a replication strategy depends on the percentage of data-items that can be stored at each machine. After scaling down, each Tier 0, 1, 2, and 3 machine has 200 Gigabytes (GB), 100 GB, 10 GB, and 2 GB storage capacity, respectively.

In the tree topology generated, each machine except those in the Tier 3 has at least one child machine in the lower tier. The bandwidth of a link connecting two machines in the different tiers is assumed as follows: 2.5 Gbit/s for a link between Tier 0 and Tier 1, 600 Mbit/s between Tier 1 and Tier 2, and 100 Mbit/s between Tier 2 and Tier 3. It should be noted that the storage space at the tiers and the link bandwidths between the tiers reduce while going downward from the Tier 0 to Tier 3 in [10], which is a trend followed here as well. Finally, the size of a data-item is randomly set to 500 Megabytes (MB), 750 MB, 1 GB, 1,25 GB, 1,5 GB, 1,75 GB, or 2 GB, and all data-items are initially stored in the Tier 0.

Heuristics: The FPH and EPP are implemented to schedule real-time data requests. Furthermore, the Fast Spread and data replacement policies are also realized in this component. The FPH and EPP are programmed to use the LRU as their dynamic data replacement policy. Two other versions of the FPH, namely FPH-1 and FPH-2, are also realized: The FPH-1 assumes the static strategy of [2], while the FPH-2 is based on that of [1] in which a data-item is kept on the intermediate machine until the respective request's deadline has been passed. Both the FPH-1 and FPH-2 rely on the LRU for the Tier 3 machines simply because their capacities are very limited to keep more than a few data-items. It should be noted that the EPP is implemented in only one version, since the performance relationship between the FPH and the other two

versions is believed to reflect the relationship between the EPP and the EPP under the data replacement policies in [1] and [2].

Request generator: The requests are submitted to the scheduler from only the Tier 3 machines. Since there are no actual data access patterns available as of now, three commonly used data access patterns are used, namely random, geometric, and Zipf [5]-[9]. The requests are assumed to come in to the Grid according to a Poisson process with a specific arrival rate. Upon its arrival, each request is associated with a randomly chosen Tier 3 destination machine and a deadline value.

Using the simulator developed, a set of simulation studies were conducted. First, a base set of results was established. In the base set, the FPH, EPP, FPH-1, and FPH-2 are evaluated in terms of the number of satisfied requests for the random, geometric, and Zipf data access patterns under the following simulation parameters: number of data-items= 100, number of requests= 1000, deadline factor= 100, and inter-arrival time of requests= 25 seconds. Later, each of these simulation parameters is individually varied to study the impact of the parameter on the performance of the respective algorithms. The results of the simulation studies are presented in Tables 1-5, where each data shown is the average of 20 simulation runs. Note that each iteration of the simulation creates a different Grid topology and request set under the given simulation parameters.

Table 1. The performance of the FPH, EPP, FPH-1, and FPH-2 under the base simulation parameters

	Base						
	Random	Geometric			Zipf		
		0.75	0.85	0.95	0.75	0.85	0.95
FPH	403	665	565	473	480	499	513
EPP	427	671	574	490	497	513	526
FPH-1	390	595	509	443	452	467	478
FPH-2	402	648	553	470	479	495	505

Table 1 shows the base set of results. Note that, for both the geometric and Zipf data access patterns, three different data-item popularity parameters (0.75, 0.85, and 0.95) related to the respective probability distribution function are used. As far as the performance is concerned, the EPP is always the best, followed by the FPH, FPH-2, and FPH-1. The EPP owes its superior performance to its ability to schedule more likely satisfiable requests along their extended paths up to their destinations usually in more than one step. The FPH performs better than both the FPH-1 and FPH-2, which reveals that a dynamic replacement policy like the LRU is better as compared to the static ones considered in this study. As expected, the FPH-1 is the worst, since it does not benefit from replication at all. The performance gain of the FPH over FPH-1 due to the replication is 8% on average. The FPH-2 enjoys the benefit of the replication to some extent and it is always superior to the FPH-1. In terms of the impact of the data access pattern on the performance, the following trend is common for all algorithms. The best results on average are obtained assuming the geometric data access pattern,

followed by the Zipf and random. The reason why the algorithms perform considerably better under the geometric and Zipf distributions is that a relatively small set of data-items are accessed many times from different machines, which boosts the efficiency of the replication. On the other hand, the longer tail of the Zipf distribution as compared to the geometric distribution is proved to have an adverse impact on the replication efficiency. Specifically, for each algorithm, the performance improvement for the geometric and Zipf data access patterns over the random one is as follows: 41% and 23% for FPH, 35% and 20% for EPP, 48% and 31% for FPH-1, and 44% and 27% for FPH-2, respectively. Based on these detailed results, it is evident that both the data replacement policy and the data access pattern have significant impact on the performance of the real-time data scheduling algorithms.

Table 2. The performance of the FPH, EPP, FPH-1, and FPH-2 under two different values for the number of data items

	No of data item= 100 (Base)			No of data item= 50		
	Random	Geometric	Zipf	Random	Geometric	Zipf
FPH	403	565	499	421	584	505
EPP	427	574	513	440	595	521
FPH-1	390	509	467	400	524	467
FPH-2	402	553	495	420	574	497

Table 2 compares the performance of the algorithms when the number of data-items is decreased from 100 to 50 while keeping the other simulation parameters unaltered. Note that, for both the geometric and Zipf data access patterns, the data-item popularity parameter of 0.85 was used for this and the following studies. Reducing the number of data-items should increase the efficiency of the replication, since the percentage of data-items that can be stored at each machine is increased. According to Table 2, all algorithms have increased their performance as expected.

Table 3. The performance of the FPH, EPP, FPH-1, and FPH-2 under two different values for the number of requests

	No of requests= 1000 (Base)			No of requests= 2000		
	Random	Geometric	Zipf	Random	Geometric	Zipf
FPH	403	565	499	689	1221	925
EPP	427	574	513	713	1241	941
FPH-1	390	509	467	657	1110	846
FPH-2	402	553	495	684	1192	908

Table 3 presents the performance of the algorithms when the number of requests is increased from 1000 to 2000 while keeping the other simulation parameters fixed. This simulation study is conducted to see how the algorithms react when the request load in the Grid is increased. In order to see the reaction, the *completion ratios*, which is the ratio between the number of satisfied requests and the number of requests, must be looked up. When the completion ratios are computed, it is seen that

all algorithms improve their performances under the geometric data access pattern, while they tend to perform worse under the random and Zipf data access patterns under heavier request load conditions.

Table 4. The performance of the FPH, EPP, FPH-1, and FPH-2 under two different values for the deadline factor

	Deadline factor= 100 (Base)			Deadline factor= 200		
	Random	Geometric	Zipf	Random	Geometric	Zipf
FPH	403	565	499	584	766	671
EPP	427	574	513	596	768	677
FPH-1	390	509	467	560	698	632
FPH-1	402	553	495	582	762	671

Table 4 shows the performance of the algorithms when the deadline factor is increased from 100 to 200 while keeping the other simulation parameters unchanged. In the simulations, increasing the deadline factor leads to relaxing the request deadlines. Thus, it is expected that all algorithms perform better when the deadline factor is 200. Indeed, Table 4 indicates that the FPH, EPP, FPH-1, and FPH-2 improve their completion ratios 38%, 35%, 38%, and 39% on average, respectively. Furthermore, the average improvements for the random, geometric, and Zipf data access patterns are 43%, 36%, and 34%, respectively.

Table 5. The performance of the FPH, EPP, FPH-1, and FPH-2 under two different values for the inter-arrival times of the requests

	Inter-arrival time= 25 sec (Base)			Inter-arrival time= 50 sec		
	Random	Geometric	Zipf	Random	Geometric	Zipf
FPH	403	565	499	606	822	706
EPP	427	574	513	621	827	718
FPH-1	390	509	467	580	745	663
FPH-2	402	553	495	603	809	700

Table 5 shows the performance of the algorithms when the request inter-arrival time is increased from 25 sec to 50 sec while keeping the other simulation parameters fixed. Relaxing the average request inter-arrival time has two consequences. First, knowing that the average data-item size is 1.25 GB based on the equally possible seven different data-item sizes, the transfer time from the Tier 0 to a Tier 3 machine is roughly 125 sec. Thus, during such a data transfer, five new requests come in to the Grid on average if the inter-arrival time is 25 sec and 2.5 if it is 50 sec. Therefore, a scheduling heuristic is put under less stress in terms of making the right scheduling decision if the inter-arrival time is lower. Second, increasing the inter-arrival time results in relaxing the requests deadlines as well because of the way followed in computing the requests deadlines. Under the combined effect of the two consequences, all algorithms have significantly improved their performances. Specifically, Table 6 indicates that the FPH, EPP, FPH-1, and FPH-2 improve their

completion ratios 45%, 43%, 46%, and 46% on average, respectively. Moreover, the average improvements for the random, geometric, and Zipf data access patterns are 49%, 46%, and 41%, respectively. When Table 4 and Table 5 are compared, it is seen that doubling the request inter-arrival time has more positive impact on the scheduling algorithm performance as compared to doubling the deadline factor.

5 Conclusions

From the results presented in the previous section, it is evident that the data replacement policy chosen and the data access pattern of the Grid users have significant impact on the performance of the real-time data scheduling heuristics. A dynamic data replacement policy like the LRU is shown to lead to the FPH to improve the Grid's performance as compared to the other two static approaches. Similar results are expected for the EPP as well. Among the data access patterns studied, all four algorithms perform better under the geometric data access pattern, followed by the Zipf and random. The long tail of the Zipf distribution is shown to adversely affect the algorithms.

References

1. Theys, M.D., Tan, M., Beck, N., Siegel, H.J., Jurczyk, M.: A Mathematical Model and Scheduling Heuristic for Satisfying Prioritized Data Requests in an Oversubscribed Communication Network. IEEE Transactions on Parallel and Distributed Systems, Vol. 11. (2000) 969-988
2. Eltayeb, M., Doğan, A., Özgüner, F.: Concurrent Scheduling: Efficient Heuristics for Online Large-Scale Data Transfers in Distributed Real-Time Environments. IEEE Transactions on Parallel and Distributed Computing, Vol. 17. (2006) 1348-1359
3. Eltayeb, M., Doğan, A., Özgüner, F.: A Path Selection-Based Algorithm for Real-time Data Staging in Grid Applications. Journal of Parallel and Distributed Computing, Vol. 65. (2005) 1318-1328
4. Chervenak, A., Foster, I., Kesselman, C., Salisbury, C., Tuecke, S.: The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. Journal of Network and Computer Applications, Vol. 23. (2000) 187-200
5. Ranganathan, K., Foster, I.: Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. Journal of Grid Computing, Vol. 1. (2003) 63-74
6. Camaron, D. G., Millar, A. P., Nicholson, C., Schiaffino, R. C., Zini, F., Stockinger, K.: Analysis of Scheduling and Replica Optimisation Strategies for Data Grids using OptorSim, Journal of Grid Computing, Vol. 2. (2004) 57-69
7. Camaron, D. G., Millar, A. P., Nicholson, C., Schiaffino, R. C., Zini, F., Stockinger, K.: Optorsim: A Simulation Tool for Scheduling and Replica Optimization in Data Grids. International Journal of High Performance Computing Applications, Vol. 17. (2003) 403-416
8. Ranganathan, K., Foster, I.: Identifying Dynamic Replication Strategies for a High-Performance Data Grid. Lecture Notes in Computer Science, Vol. 2242. (2001) 75-86
9. Lamehamedi, H., Shentu, Z., Szymanski, B., Deelman, E.: Simulation of Dynamic Data Replication Strategies in Data Grids. Heterogeneous Computing Workshop. (2003) 100b
10. GridPP Collaboration: GridPP: Development of the UK Computing Grid for Particle Physics. Journal of Physics G: Nuclear and Particle Physics, 32 N1-N20. (2006)

Multi-site Scheduling with Multiple Job Reservations and Forecasting Methods

Maria A. Ioannidou and Helen D. Karatza

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{vakis, karatza}@csd.auth.gr

Abstract. Most previous research on job scheduling for multi-site distributed systems does not take into consideration behavioral trends when applying a scheduling method. In this paper, we address the scheduling of parallel jobs in a multi-site environment, where each site has a homogeneous cluster of non-dedicated processors where users submit jobs to be executed locally, while at the same time, external parallel jobs are submitted to a meta-scheduler. We use collected load data to model the performance trends that each site exhibits in order to predict load values via time-series analysis and then perform scheduling based on the predicted values.

1 Introduction

A multi-site system typically consists of different clusters of computers which most probably have completely different characteristics either in computational capabilities, network communication or simply the number of resources available. Furthermore, in the case where these systems are not dedicated, that is, there are users submitting jobs for local execution during the day, their behavior changes even in the case where their performance characteristics stay the same. In those systems, an important issue that has an obvious impact on performance is the local users' job submission scheme. Analyzed data from several supercomputer centers show that the resource demand in such a system exhibits a wide variance from maximum to minimum capacity of the system. Since most users tend to request resources that follow a timely pattern, apparently it would be useful to identify the trends that lie in users' job submissions and use them to make an efficient prediction model for the system's behavior.

The multiple submissions have proven to offer better turnaround times in an heterogeneous context [1], but there is a considerable overhead involved in maintaining a large number of reservations due to network latency bound. We use time-series prediction models to minimize the number K of selected sites and therefore reduce the network communication overhead improving the overall turnaround time.

Jobs submitted to the meta-scheduler consist of parallel tasks that are scheduled to execute concurrently on a set of processors. This type of resource management is called "gang scheduling" or "co-scheduling". There are several techniques to implement gang scheduling from which the simplest of all is considered to be *local scheduling* (uncoordinated), where no measures are taken to coschedule processes in a gang [2].

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 894 – 903, 2006.
© Springer-Verlag Berlin Heidelberg 2006

The paper is organized as follows. In Section 2, we provided some background information about heterogeneous job scheduling, time-series analysis and prediction models as well as related previous work. Section 3 provides information about the simulation model used for this study and describes the scheduling strategies that we use. Model implementation along with the results of the simulation experiments are presented and analyzed in section 4. Section 5 is the conclusion and provides suggestions for further research and the sixth section is references.

2 Related Work

There has been a considerable body of work on scheduling parallel jobs either in homogeneous or in heterogeneous context. [3] discusses the advantages of multi-site scheduling. As examined in [1] better turnaround times can be achieved with multiple requests to a number of sites and aggressive or conservative backfilling scheduling strategies. In [4] the authors use multiple requests with priorities for local jobs, in [5] task duplication is used, and [6] uses adaptive multi-site scheduling that via a simple decision rule decides whether to use or not to use multi-site scheduling. So the notion of submitting simultaneously the same job to more than one processing unit is proven to have improved results. [7] proposes an informed-search algorithm for restricted complexity problems, [8] utilizes probabilities of information received by sensors with the Bayesian approach, [9] uses stochastic scheduling to achieve good and predictable performance, [10] uses past information in a multi-processor system and shows the effectiveness of choosing randomly two processors, and [11] models execution time based on measurement results of various configurations. Therefore, there has been some effort to use the information in hand from actual use of the system, in order to facilitate our scheduling decisions in the future.

We use time series as a forecasting methodology. The intrinsic nature of time series is that observations are dependent or correlated, and the order of the observations is therefore important. Time series analysis is a suitable technique to successfully model a system minimizing forecast errors as much as possible, as is done in [12].

3 Model and Methodology

3.1 System and Workload Models

We consider a multi-site grid consisting by N sites with $P_i = 2^{i+2}$ processors per site i, where $i = 0, 1, \ldots, N\text{-}1$. Each site is considered as a homogeneous cluster with identical processors, while the whole system of the N sites is a heterogeneous one, since $P_i \neq P_j$ when $i \neq j$ for $i, j = 0, 1, \ldots, N\text{-}1$. A job x consists of m_x tasks where $1 \leq m_x \leq \max(P_i)$, $i = 0, 1,\ldots, N\text{-}1$. This means that all jobs can run to at least one of the simulated sites. In this paper, we do not consider the scheduling of a single job across multiple sites. When the number of parallel tasks in a job is greater than the number of processors in a site s, then s is excluded from the site selection set and the scheduler has to substitute it with one or the rest of the sites where $m_x \leq P_i$, $i = 0, 1,\ldots, s\text{-}1, s\text{+}1,\ldots, N\text{-}1$.

Jobs are processed in arrival order by the global scheduler and at the same time, there are jobs submitted by users locally thus contributing to the load of each site. Those jobs are executed in the site where the submission is taking place, so the global scheduler does not have to make any site selection decision about them. Both types of jobs enter the same waiting queues when the site is selected.

Each job begins execution only when enough idle processors are available to meet its needs. When a job terminates execution, all processors assigned to it are reclaimed.

We assume that there is no correlation between job size and task service demand. For example, a small job may have a long service time. The number of jobs that can be processed in parallel depends on job sizes and on the scheduling policy applied. We also assume that job sizes are uniformly distributed over the range $[1..2^N]$. Since $P_i = 2^{i+2}$, there are always at least two sites where a job may find sufficient resources, even if it has the maximum size.

The jobs that are submitted to the global scheduler are considered to have inter-arrival times that follow the exponential distribution with a mean of $1/\lambda$, while the service demands of tasks are also exponentially distributed with a mean of $1/\mu$.

At the same time jobs arrive at the meta-scheduler, users submit their own jobs to each of the N sites, as a Poisson stream of rate λ_l. For each job the destination site is predetermined and for this simulation experiments it is chosen randomly. These jobs are also scheduled for execution according to the local scheduling policy of the particular site. We consider those jobs to be sequential in nature, which means that they do not require more than one processor for their execution. A similar scheme, with parallel and sequential jobs submitted simultaneously is examined in [13]. They do not contribute to the performance metrics other than as a load variance factor for the computations we need to make for the parallel jobs performance evaluation. This means, that although the users jobs exist as an additional load factor for the sites' queues and processors, we are only interested in the average turnaround time and average slowdown of the parallel jobs.

3.2 Scheduling Algorithms

First, we consider the random choice of K sites where each job is automatically submitted. It has been shown that choosing $K = 2$ instead of one random destination yields an exponential improvement and from then on, the improvement is by just a small factor, if not negative due to network latencies from the multiple requests sent back and forth [14], [1], [2].

Then we follow the greedy scheme of assigning each job to the site with the lowest instantaneous load. The instantaneous load is considered to be the ratio of the total remaining processor-runtime product for all jobs (either queued or running at that site) to the number of processors at the site [1]. When a job is submitted to the meta-scheduler, a computation of each site's instantaneous load is performed, the K least loaded sited are chosen and the job is submitted to all K sites at the same time.

The third option is the use of predicted values for the load and then assign the jobs to the least predicted loaded sites. To construct the prediction model, we calculate the instantaneous loads of each site for an adequate number of jobs. We use the previous data to perform time series analysis producing a load prediction model via Single Exponential Smoothing. The particular scheme weights past observations with exponentially

decreasing weights to forecast future values. Its characteristic is that it gives to recent observations relatively more weight in forecasting than the older observations.

The basic equation for the single exponential smoothing model is

$$S_t = \alpha y_{t-1} + (1-\alpha) S_{t-1}, \quad t \geq 3, \quad 0 < \alpha \leq 1 \ . \tag{1}$$

This can be written as:

$$S_{t+1} = S_t + \alpha(\varepsilon_t) \ . \tag{2}$$

where ε_t is the forecast error (actual – forecast) for period *t*.

So when the load changes due to these submissions, the forecasting model will take into account most recent load values thus minimizing prediction error. The main parameter for this scheme is the smoothing constant which is a meter of how fast we "dump" old values in benefit of new ones. Since $0 < \alpha \leq 1$ we choose the value that minimizes the MSE from 0.1 to 0.9. Our system's variability factor is the different number of processors per site, therefore the smoothing constant could variate between specific values according to the way the job sizes are distributed through time, in order to provide a more sufficient forecasting model.

Three algorithms are used for the scheduling of jobs at the sites, Adaptive First Come First Serve (AFCFS), Largest Gang First (LGF) and Aggressive Backfilling (AgBF).

3.3 Performance Metrics

Response Time r_j of a job *j* the time spent in the site's queue plus the job service time. *Slowdown* s_j of a job *j* is the job's response time divided by the job's service (run) time. It measures the delay of a job against it's actual runtime.

Utilization u_j of a site *j* is the fraction of time when a site's resources are used to the total simulation time. But since per each time unit there is a different portion of the site's resources being utilized, we need to insert to this definition the number of processors used per time unit [15]. So, if we partition the total simulation time in time intervals, where each interval begins each time the number of busy processors changes, the utilization of site *i* is defined as:

$$U_i = \frac{\sum_{j=1}^{T} b_j \cdot \Delta t_j}{P_i \sum_{j=1}^{T} \Delta t_j} \ , \tag{3}$$

where b_j are the busy processors during the time Δt_j, P_i the number of the site's processors and *T* is the number of time intervals during which the number of busy processors is unchanged.

The following metrics were used for perfomance evaluation:

The Mean Response (Turnaround) Time $\overline{RT}$:

$$\overline{RT} = \frac{1}{n} \sum_{j=1}^{n} r_j \ , \tag{4}$$

where n is the total number of processed jobs.

The Mean Slowdown $\overline{SLD}$:

$$\overline{SLD} = \frac{1}{n}\sum_{j=1}^{n} s_j \ , \tag{5}$$

where n is the total number of processed jobs.

The Mean Utilization $\overline{U}$:

$$\overline{U} = \frac{\sum_{j=1}^{T} b_j \cdot \Delta t_j}{\sum_{i=0}^{N} P_i \sum_{i\ j=1}^{T} \Delta t_j} \ . \tag{6}$$

4 Simulation Results and Discussion

4.1 Model Implementation and Input Parameters

We first implemented the system's model selecting K=2 sites randomly from the available N=4 sites with the restriction that jobs are submitted only to sites where they can find enough resources. That is, the number of tasks of each job is always smaller than the total processors of the chosen sites. The maximum number of tasks per job is 16, while the maximum number of processors per site is $P_i = 2^{i+2}$, $i = 0,..3$, so every job that is submitted to the meta-scheduler is able to find at least two sites with sufficient resources.

The queueing model is simulated with discrete event simulation modeling [16], using the independent replication method. For each set of workload parameters we run 30 replications of the simulation with different seeds of random numbers and for 40000 served jobs in each replication. For every mean value, a 95% confidence interval is evaluated. All confidence intervals are less than 5% of the mean values. We set the mean service time: $1/\mu$ = 1. The number of tasks is as we mentioned uniformly distributed in the interval [1, 16], so the average gang size is $\frac{1+16}{2} = 8.5$. The total processors in all sites are 60, therefore when all of them are busy, the average number of jobs that can be served per time unit is 7.059. So, we have to choose a λ as job arrival rate such that it hold the conditions $\frac{1}{\lambda} > \frac{1}{7.059} \simeq 0.142$, i.e. the sites queues will not be saturated. But we also have to take under consideration the job submissions by the local users, that contribute towards the system's saturation. After experimental runs with various values of $1/\lambda$ we chose 0.172 as the smallest mean inter-arrival time for the experiments, and the rest of the values that we used are: 0.178, 0.185, 0.192, 0.2.

For the users jobs we have to take under consideration the fact that in order for their presence in the system to have an impact on the system's behaviour, we have to

set their arrival rate λ_1 so that their contribution to the system's load status remains significant without overloading it.

Since the total number of processors is 60 and the average job size is 1, then $\frac{1}{\lambda} > \frac{1}{60} \simeq 0.017$, which means that based on the fact that the jobs are uniformly distributed among the sites, with an average of 15 jobs/time-unit per site, the system is saturated. But because the sites potential of serving jobs is different due to their different processor numbers, we have to take under consideration the fact that we cannot estimate $1/\lambda_1$ based to the system's total number of processors, because that way we would overwhelm site *0* ($P_0 = 4$) with too many jobs from the users, making it impossible for the gangs to use it. So for site 0, the number 3.75 jobs/time-unit is enough to become saturated. We use the formula

$$\frac{1}{\lambda_1} = \frac{N \cdot P_0}{\sum_{i=0}^{N-1} P_i - \lambda} \tag{7}$$

to produce λ_1 as a function of λ in order to avoid overwhelming the smaller sites with bulk arrivals of users jobs. Therefore, the corresponding values of $1/\lambda_1$ for the previous values of $1/\lambda$ are: 0.295, 0.294, 0.293, 0.292, 0.291.

4.2 Performance Analysis

Figure 1 shows the simulation results for the system's mean utilization, mean response time and mean slowdown for both the three scheduling algorithms. It is clear that LGF outperforms AFCFS, as in [17] and [18] also, but AgBF has significantly better results in terms of mean slowdown and mean response time.

We performed the load calculations for different values of α. The best value is the one that minimizes the Mean Squared Error (MSE) of actual values vs the predicted ones [19]. If $\alpha = 0.9$ it means that the mostly weighted values are the most recent. Based on the results, if we don't want to fully reject past values we should choose a smoothing constant equal to 0.7, where the MSE is stabilized.

The predicted values, follow the actual ones very closely, which means the prediction error for all cases is very small and therefore the prediction model is trustworthy. The error is higher for the first values because the model uses them to adjust better to the subsequent values. The error variation seems to be higher for the case of site 0, which is due to its small capacity (only 4 processors) that produces high variation to its utilization. The site may be full with a single job at a certain time and the next moment with only one processor occupied by a user's job. The higher the variation, the more difficult it gets for the model to "smooth" the predicted values.

There is a cost when the job contacts all sites to see whether it should start, based on the current policy and then, when a job is ready to start, there is the cost of notifying all the other sites where it was submitted, to cancel their reservation. This cost is in terms of network latency and involves at least 3(K-1) messages. There is also the cost from the job's transfer to each of the selected sites, which is in terms of network bandwidth. It is obvious that this overhead is a function of K, and that the smaller K is, the smaller the communication cost.

But the multiple reservation scheme benefits more as K increases, so by using the prediction model, we reject a number of reservations, thus reducing network communication between sites without having to choose a smaller K. We use as criterion, in order to choose which reservation should be rejected, the standard deviation between sites' loads. When the load exceeds the standard deviation of the mean loads, the site is rejected.

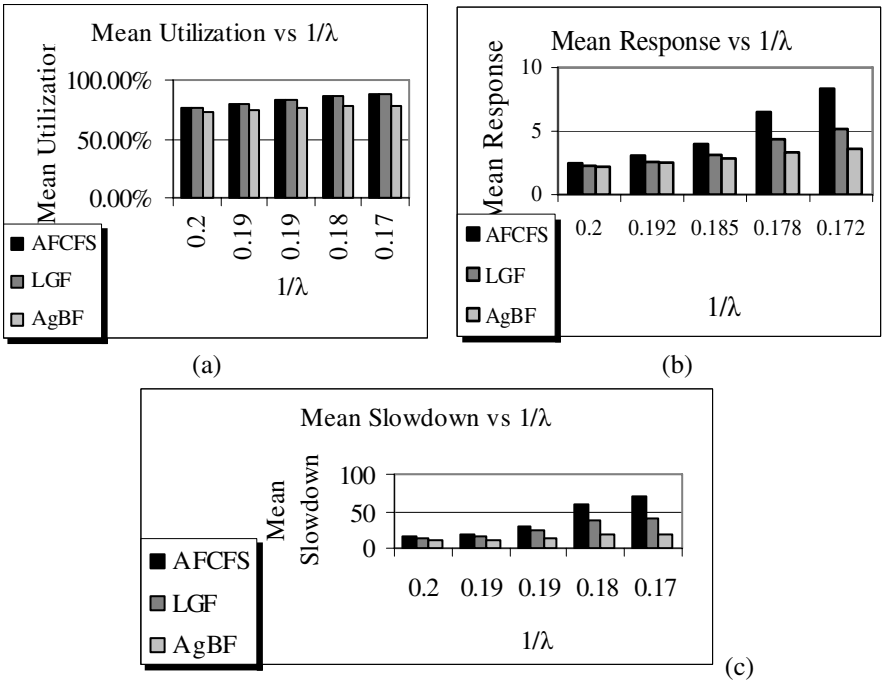


Fig. 1. Mean utilization (*a*), response (*b*) and slowdown (*c*) for different values of λ for the three scheduling algorithms AFCFS, LGF and AgBF

The strategy that seems to benefit most from the use of the instantaneous load as a site selection criterion is LGF with significant improvement in average response and slowdown. This is probably because of the smaller jobs which get to choose more wisely their destination and avoid getting stuck behind many large jobs. The smaller jobs also have the advantage of more available choices since they don't usually have to discard a site because of the insufficient number of processors. Large jobs have fewer suitable sites to select and so when many large jobs concentrate to relatively large sites, the smaller jobs have the opportunity to balance the load by selecting the "smaller" sites where they won't be delayed behind larger jobs.

We repeat the above simulation runs using the predicted values this time in selecting the K sites. The cases where the model fails to make the right choices, can be considered statistically insignificant for a large set of jobs.

The variation for the prediction errors, concerns only the first 1300 job arrivals, where the error is higher, because the model uses the first values to adjust itself. After

a while the model works with almost zero error. We actually used 80000 jobs for the simulation experiments. With the use of a simple linear formula as communication overhead model, where the overhead is a function of K, we calculate it with the values for K that are produced via the use of the prediction scheme.

We present the cases of the AgBF and LGF algorithms as scheduling policies at the sites. The results are shown in Figure 2.

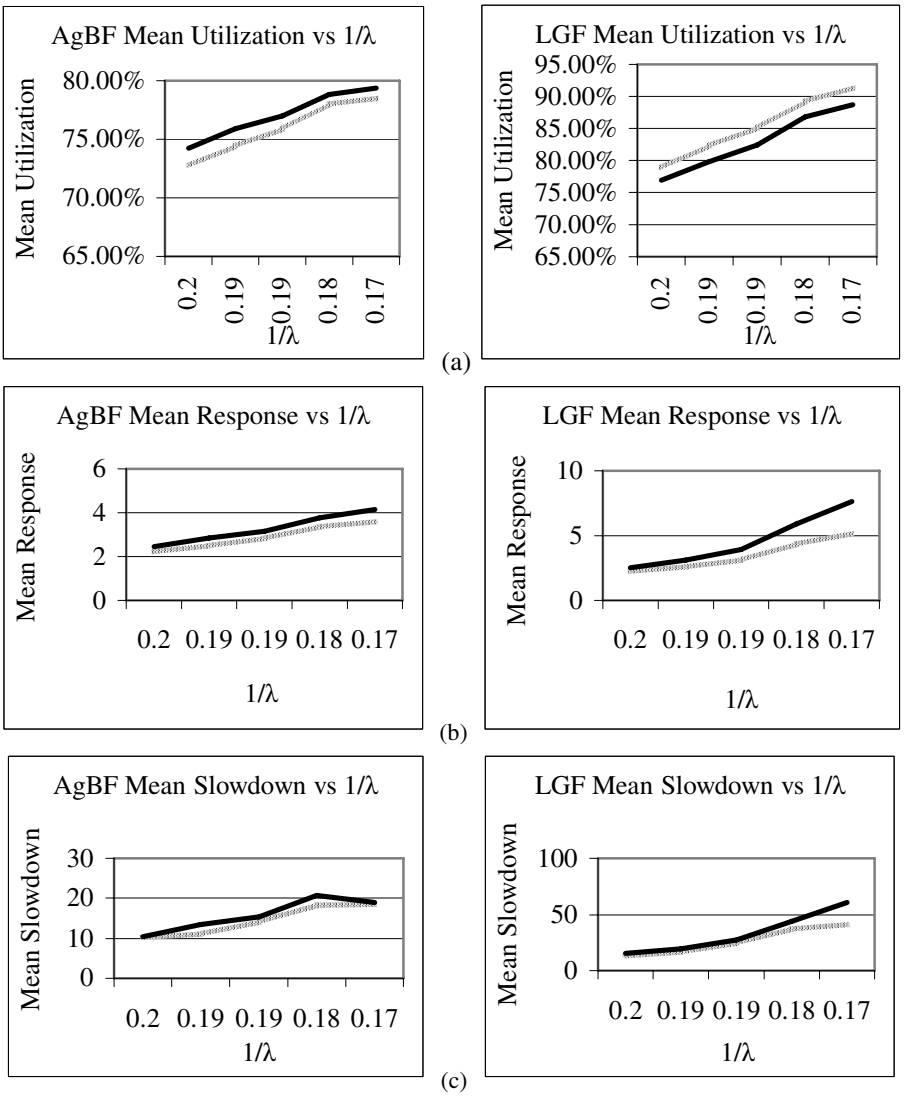


Fig. 2. (*a*) Mean Utilization, (*b*) Mean Response and (*c*) Mean Slowdown for AgBF (*left*) and LGF (*right*) with (*black line*) and without (*grey line*) the use of the prediction model

The results show a slight degrade in terms of mean utilization, a 1.51% for AgBF and a 2.85% for LGF in average, which was expected since with greater latencies, there are fewer jobs available to fit in the processors at the same time. In terms of mean slowdown, there is an average improvement of a 8.87% for AgBF and a 17.19% for LGF, but the best results are for the mean response, where there is an average improvement of 11.17% for AgBF and 21.25% for LGF. The LGF algorithm is once again the one that benefits the most even though its performance in general is worse than the AgBF's. This is due to the fact that when the larger jobs delay because of communication overhead they make it even harder for the smaller jobs to find a suitable set of sites to reserve and the performance of LGF is based on the relative behaviour of small versus large jobs. When the prediction scheme is used, the messages between sites are reduced, thus the communication cost is reduced and for the LGF this is even more prominent because there are fewer sites available for selection anyway, so K is minimized and the algorithm performs better.

5 Conclusion

The use of predicted load values results in lowering the communication overhead that comes with the multiple reservation scheme, by reducing the number of the selected sites where the job is to be reserved. Most users follow a certain behavioural pattern in the jobs they submit. Time series analysis exploits this pattern to create a prediction model of the system's status in terms of load.

We plan to extend the research in cases where local submissions are not just uniformly consistent, but show a trend or periodicity, demanding appropriate adjustments to the corresponding prediction model and to test the model's reliability with data from an actual grid.

References

1. G. Sabin, R. Kettimuthu, A. Rajan and P. Sadayappan, "Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment", Proceedings of the 8th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), 2003.
2. Strazdins, P. and J. Uhlmann, "A Comparison of Local and Gang Scheduling on a Beowulf Cluster". In Proceedings of 2004 IEEE International Conference on Cluster Computing (San Diego, CA, Sept. 20-23). IEEE Computer Society, Los Alamitos, CA, 55-62.
3. C. Ernemann, V. Hamscher, U. Schwiegelshohn and R. Yahyapour, "On advantages of grid computing for parallel job scheduling", Proceedings of 2nd IEEE International Symposium on Cluster Computing and the Grid (CC-GRID 2002), Berlin, Germany, pp. 39-46, 2002.
4. V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests", Proc. of 11-th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), July 2002.
5. Yu-Kwong Kwok, "On exploiting Heterogeneity for cluster based parallel multithreading using task duplication", The Journal of Supercomputing, May 2003, Volume 25, Issue 1, pp. 63 - 72.

6. C. Ernemann, V. Hamscher, R. Yahyapour and A. Streit, "Enhanced algorithms for multi-site scheduling", Proceedings of 3rd International Workshop Grid 2002, in conjunction with Supercomputing 2002, Baltimore, MD, USA, pp. 219-231, November 2002.
7. M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems", IEEE Concurrency, July 1998, Volume 6 , Issue 3 pp. 42 – 51.
8. R. Real, A. Yamin, L. da Silva, G. Frainer, I.Augustin, J. Barbosa and C. Geyer, "Resource scheduling on grid: handling uncertainty", *Proceedings of the Fourth International Workshop on Grid Computing (GRID'03) IEEE,* 2003.
9. J. M. Schopf, F. Berman, "Stochastic Scheduling", Conference on High Performance Networking and Computing, Proceedings of the 1999 ACM/IEEE conference on Supercomputing, Article No. 48.
10. M. Mitzenmacher, "How useful is old information", IEEE Transactions on Parallel and Distributed Systems, January 2000, Volume 11, Issue 1 pp. 6–20.
11. Y. Kishimoto and S. Ichikawa, "An execution-time estimation model for heterogeneous clusters", Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), IEEE 2004.
12. Peter Hoogenboom, Jay Lepreau, "Computer System Performance Proble Detection Using Time Series Models", Proceedings of the USENIX Summer 1993 Technical Conference, Cincinnati, Ohio, June 21-25, 1993.
13. H.D. Karatza and R.C. Hilzer. "Scheduling Sequential Jobs and Gangs in a Distributed Server System". Proceedings of the 5th EUROSIM Congress on Modelling and Simulation (Special Session on Modelling and Simulation of Distributed Systems and Networks), September 06-10, 2004, ESIEE Paris, Scit Descartes, Marne la Valle´, FRANCE, EUROSIM-FRANCOSIM-ARGESIM, 2004, pp. 17-22.
14. Michael Mitzenmacher, Berhold Vocking, "The Asymptotics of Selecting the Shortest of Two, Improved", Allerton99.
15. H.D. Karatza, "Performance Analysis of Gang Scheduling in a Partitionable Parallel System". Proceedings of 20th European Conference on Modeling and Simulation. 28-31 May 2006, Bonn, Sankt Augustin, Germany, pp. 699-704.
16. Law A. and Kelton D. 1991, *Simulation Modeling and Analysis*. 2nd Ed., McGraw-Hill, Inc, New York, USA.
17. H.D. Karatza. "Gang Scheduling in a Distributed System under Processor Failures and Time-varying Gang Size", Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS '03), IEEE Computer Society Press, May 28-30 2003, San Juan, Puerto Rico, pp. 330-336.
18. H.D. Karatza, "Scheduling Gangs in a Distributed System", International Journal of Simulation: Systems, Science Technology, UK Simulation Society, January 2006, Volume 7, issue no: 1, pp. 15-22.
19. William W. S. Wei 1994, *Time Series Analysis - Univariate and Multivariate Methods*, 1st Ed. Reprinted, Addison-Wesley, Inc, Redwood City, California, USA.

Evaluating the Dynamic Behaviour of *PROSA* P2P Network

Vincenza Carchiolo, Michele Malgeri, Giuseppe Mangioni, and Vincenzo Nicosia

Dipartimento di Ingegneria Informatica e delle Telecomunicazioni
Facoltà di Ingegneria – Università di Catania
Viale A. Doria 6 – 95100 Catania (Italy)
{car, malgeri, gmangioni, vnicosia}@diit.unict.it

Abstract. In this paper we present and simulate a new self–organising algorithm for P2P unstructured networks inspired by human relationships. This algorithm, called ***PROSA*** , tries to simulate the evolution of human relationships from simple acquaintance to friendship and partnership. Our target is to obtain a self–reconfiguring P2P system which possesses some of the desirable features of social communities. The most useful property of many natural social communities is that of beeing "small–worlds", since in a small–world network queries usually require a small amount of "hops" to walk from a source to a destination peer. We show that ***PROSA*** naturally evolves into a small–world network, with an high clustering coefficient and a relly short average path length.

1 Introduction

The way social contacts and relationships are arranged, how they evolve and how they end, is matter for psychologists and social scientists research. Nevertheless some studies about social groups and their connections reveal that a "social network", i.e. the network of relationships among people from simple acquaintance to friendship, has many interesting properties that can be exploied in a real–world P2P structure.

The Milgram experiment of 1966 [6] showed that a message from a "source" to a "destination" person can be delivered by forwarding it step–by–step to just one of the related people, in the direction of the destination. In practise Milgram asked to sixty people located in Kansas to send a letter to a specified person located in Cambridge. The participants could just pass the letter to personal acquaintances, hand–by–hand. About one quarter of the total number of letters were delivered to the destination person, and Milgram found that the mean number of "hops", i.e. the number of persons involved in each delivery, was about six. This experiment opened the research in the field of "small–world" networks [9].

The small–world property seems to be a characteristic of many human communities, such as mathematicians, actors, scientists. A small–world arises almost naturally whenever social contacts among people are involved: many researchers are trying to understand the reasons of this behaviour. In this work we're not interested in answering this question. Our target is just to develop a P2P system using rules and concepts inspired by human behaviours and relationships dynamics.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 904–915, 2006.
© Springer-Verlag Berlin Heidelberg 2006

In a social network there are several kind of social links among people. We can identify "acquaintance–links" and "semantic–links": the former expresses a simple "acquantaince" among people; the latter requires at least an acquaintance–link plus some additional information about interests, culture, abilities, knowledge etc. In our life semantic–links arise almost naturally. You need no great effort to establish a semantic–link with somebody: you have just to share a knowledge field or a passion or simply an interest with a person and meet him in some circumstances, have a talk with him and no more. Once you know somebody shares a certain knowledge or passion with you, a semantic–link in that field with that person is established and you're ready to use that link the next time you need information, help, assistance or collaboration in that field.

In real life we massively use semantic–links to speed up information retrieval. For example if a car vendor wants information about Linux, he asks his nephew, who is studying Computer Science at University, but doesn't ask his wife since she is a biologist and she does not like computers. Note that both of them (this nephew and his wife) are "semantic–links", but they belong to two different semantic fields. His nephew, who doesn't know anything about Linux, will ask to one of his colleagues of the Operating Systems course, who is famous as being a Linux guru and can give him the required information: using semantic–links a car vendor reaches a Linux guru in just two hops. If the car vendor in our example doesn't have a nephew studying Computer Science, he asks to a friend (acquaintance–link) at random, hoping somebody knows what Linux is. Our daily experience says that, at the end, he will find somebody who can help him gathering information about Linux.

The same mechanism that allowed Milgram's letter to be correctly delivered from Kansas to Cambridge in just six hops is exploited in the given example: small–world characteristic of a network allows efficient information retrieval. This is how small–world networks work.

In this paper we introduce a P2P structure, named ***PROSA*** [2], in which semantic proximity of resources is mapped onto topological proximity of peers. ***PROSA*** is inspired by social relationships and their dynamics, since social networks characteristics can be exploited to optimise query forwarding and answering. ***PROSA*** uses a self–organising algorithm that dinamically links peers sharing similar knowledge and resources, putting them into high clustered and self–structured "semantic groups". To validate the proposed algorithm we developed a functional simulator and we used it to show that ***PROSA*** really evolves into a small–world network.

The paper is organised as follows: Section 2 is a short survey about current work in the field of P2P resource retrieval; in Section 3 we discuss our proposal; in Section4 we show simulation results and finally Section 5 presents a plan for future work.

2 Related Work

In the last years the interest on overlay networks has increased, mainly because bandwidth, computing power and cheapness of personal computers allow to implement such kind of "logic" networks. Examples of overlay networks include Gnutella, Freenet [3], CAN [5], Tapestry [10]. Each of them focuses on a particular aspect of P2P computing: Gnutella is totally unstructured, Freenet is practically anonymous, CAN is search–efficient and so on.

Few P2P structures proposed till now face the problem of efficient resources retrieval. In particular one of the more desirable feature in a P2P network is the possibility to perform query based on semantic resource description. Semantic queries are interesting because they are similar to the natural way a user describe concepts.

In unstructured networks, such as Gnutella, semantic query for resource can be performed, but for each request most part of the network is flooded, and there are no response guarantees either if the requested resource is present ([4]). In networks organised as Distributed Hash Tables (DHT) [3][5][10] semantic queries are not allowed, since resources are described by a certain hash of their content or description, so no "semantic proximity" can be neither defined nor used to discover them. Some recent works [1][11] proposed to organise a P2P network in semantic groups of "similar" peers, to facilitate resource search and retrieval based on semantic queries. In particular in SETS [1] the network is split in semantic areas by a super–peer which also maintains a table of groups centroids; a centroid represents the "topic" of a given area. The main drawback of SETS is the introduction of a network manager, which represents a single point of fault. In GES [11] peers maintains two sets of links to other peers: semantic–links and random–links. Queries for resources are first forwarded to a so–called "semantic–target", which is the first peer that can answer the query, and then flooded to this peer neighbours (the semantic group).

3 *PROSA*

Our target is to create a P2P network based on acquaintance– and semantic–links, where peers join the network in a way similar to a "birth", then achieve more links to other peers according to the social model, i.e. by linking (semantically) with peers which have similar interests, culture, hobbies, works and so on, and maintaining a certain number of "random" acquaintances. In P2P networks the culture or knowledge of a peer is represented by the resources (documents) it shares with other peers. On the other hand, different types of "links" among peers simulate acquaintances and semantic–links. To implement such a model it is necessary to have:

- A system to model knowledge, culture, interests etc...
- A self–organising network management algorithm

3.1 Modelling Knowledge

In ***PROSA*** , knowledge (each resource) is represented using the Vector Space Model (VSM) [8]. In this approach each document is represented by a state–vector of (stemmed) terms called Document Vector (DV); each term in the vector is assigned a weight based on the relevance of the term itself inside the document. This weight is calculated using a modified version of TF–IDF [7] schema, as follows:

$$w_t = 1 + \log(f_t)$$

where f_t is the term frequency into the document. It has been proved [8] that this way of calculating relevance is a good approximation of TF–IDF ranking schema. The VSM

representation of a document is necessary to calculate the relevance of a document with respect to a certain query. We model a query by means of a so–called Query Vector (QV), that is the VSM representation of the query itself. Since both documents and queries are represented by state–vectors, we define the relevance of a document (D) with respect to a given query (Q) as follows:

$$r(D,Q) = \sum_{t \in D \cap Q} w_{t,D} \cdot w_{t,Q} \tag{1}$$

Using VSM we obtain also a compact description of a peer knowledge. This description is called "Peer-Vector" (PV), and is computed as follows:

- For each document hosted by the peer, the frequencies of terms it contains are computed ($F_{t,D}$).
- Terms frequencies for different documents are summed together, obtaining overall frequency for each term:
$$F_t = \sum_t F_{t,D}$$
- Then a weight is computed for each term, using:
$$w_t = 1 + \log(F_t)$$
- Finally all weights are put into a state–vector and the vector is normalised.

The obtained PV is a sort of "snapshot" of the peer knowledge, since it contains information about the relevant terms of the documents it shares.

The relevance of a peer (P) with respect to a given query (Q) is defined as follows:

$$r(P,Q) = \sum_{t \in P \cap Q} w_{t,P} \cdot w_{t,Q}$$

This relevance is used by the ***PROSA*** query routing algorithm. It is worth noting that a high relevance between a QV and a PV means that probably the given peer has documents that can match the query.

3.2 Network Management Algorithm

As stated above, relationships among people are usually based on similarities in interests, culture, hobbies, knowledge and so on. And usually these kind of links evolve from simple "acquaintance–links" to what we called "semantic–links".

To implement this behaviour three types of links have been introduced:

- Acquaintance–Link (AL)
- Temporary Semantic–Link (TSL)
- Full Semantic–Link (FSL)

TSLs represent relationships based on a partial knowledge of a peer. They are usually stronger than ALs and weaker than FSLs.

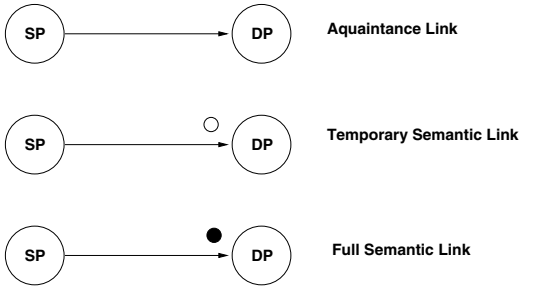


Fig. 1. Link types

Since usually relationships are not symmetric, it is necessary to specify what are the source peer (SP) and destination peer (DP) of a link. Figure 1 shows the representations for the three different types of links.

To efficiently use the appropriate link in any given situation, each peer maintains a list of known peers, that we call Peer List (PL). It is a finite list of links divided into three parts: the first one contains FSLs, the second one contains TSLs and the third contains ALs. The size of each portion is an algorithm parameter. Note that FSLs represent "stable" connections (parents, relatives), TSLs are similar to FSL but are not so strong (friends, colleagues), and finally AL are really weak links.

Joining. The case of a node that wants to join an existing ***PROSA*** network is similar to the birth of a child. At the beginning of his life a child "knows" just a couple of people (his parents). A new peer which wants to join, just searches other peers (for example using broadcasting, or by selecting them from a list of peer that are supposed to be up, as in Freenet or Gnutella) and adds some of them in his PL as ALs. These are ALs because a new peer doesn't know anything about its "relatives" until he doesn't make query to them for resources. This behaviour is quite easy to understand: when a baby comes to life he doesn't know anything about his parents. He doesn't know his father's job, neither that is mother is a biologist. The joining phase is represented in figure 2, where "N" is the new peer; N chose some other peers (P) at random as initial ALs.

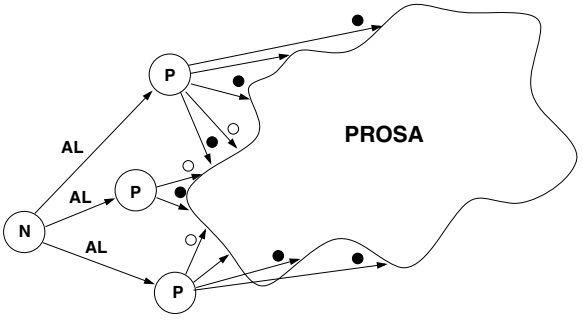


Fig. 2. A new node joining ***PROSA***

Updating. In ***PROSA*** FSLs dynamics are strictly related to queries. When a user of ***PROSA*** requires a resource, he performs a query and specifies a certain number of results he wants to obtain. The relevance of the query with respect to the resources hosted by the user's peer is first evaluated, using equation 1. If none of the hosted resources has a sufficient relevance with respect to the query, the query has to be forwarded to other peers. The mechanism is quite simple:

- A query message containing the QV, a (possible) unique QueryID, the source address and the required number of results is built.
- If the peer has neither FSLs nor TSL, i.e. it has just AL, the query message is forwarded to one link at random.
- Otherwise, the peer computes the relevance between the query and each entry of his Peers–List.
- It selects the link with a higher relevance, if it exists, and forwards the query message to it.

When a peer receives a query forwarded by another peer, it first updates its PL. If the requesting peer is an unknown peer, a new TSL to that peer is added in the PL, and the QV becomes the corresponding Temporary Peer Vector (TPV). If the requesting peer is a TSL for the peer that receives the query, the corresponding TPV in the list is updated, adding the received QV and normalising the result. If the requesting peer is a FSL, its PV is in the PL yet, and no updates are necessary.

After PL update, the relevance of the query and the peer resources is computed. There are three possible cases:

- None of the hosted documents has a sufficient relevance. In this case the query is forwarded to another peer, using the same mechanism used by the forwarder peer. The query message is not modified.
- The peer has a certain number of relevant documents, but they are not enough to full-fill the request. In this case a response message is sent to the requester peer, specifying the number of matching documents and the corresponding relevance. The message query is forwarded to all the links in the PL whose relevance with the query is higher than a given threshold (semantic flooding). The number of matched resources is subtracted from the number of total requested documents before forwarding.

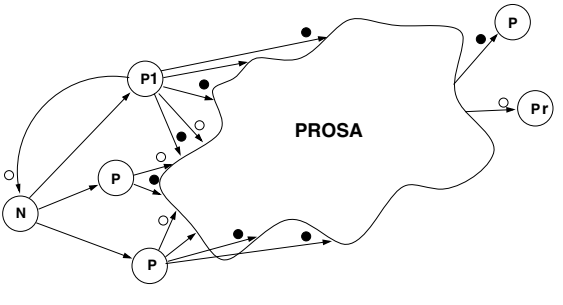


Fig. 3. Query forwarding: new TSL arise

- The peer has sufficient relevant documents to full-fill the request. In this case a result message is sent to the requesting peer and the query is no more forwarded.

This situation is showed in figure 3, where peer "N" forwards a query to one of his ALs randomly chosen, since it has niether TSLs nor FSLs. In our example the chosen peer is "P1". As soon as P1 receives the QV, it automatically establish a TSL with N (see figure 3) and then it forwards the query if needed.

When the requesting peer receives a response message it presents the results to the user. If the user decides to download a certain resource from another peer, the requesting peer contacts the peer owning that resource and asks it for download. If download is accepted, the resource is sent to the requesting peer, together with the Peer Vector of the serving peer. This case is illustrated in figure 4, where peer "N" received a response from peer "Pr" and decided to download the corresponding resource. Note that Pr established a TSL with N, because it received a QV from it, and N established a FSL with Pr, because it successfully received a resource from it.

4 *PROSA* Simulations and Results

In order to validate ***PROSA*** dynamics and effectiveness, we developed a simple functional simulator in Python. The three "phases" of peers life (joining, querying and leaving ***PROSA***) are represented by different "behaviours" and triggered by events off-line generated by an event-generator. The choice of data sets is of the most importance in order to obtain consistent and relevant simulation results. To simulate ***PROSA*** functionalities, we chose to use a data set composed by scientific articles from two different fields: Maths and Philosophy. Maths articles comes from "Journal of American Mathematical Society", "Transactions of the American Mathematical Society" and "Proceedings of the American Mathematical Society", for a total amount of 740 articles. Philosophy articles comes from "Journal of Social Philosophy", "Journal of Political Philosophy",

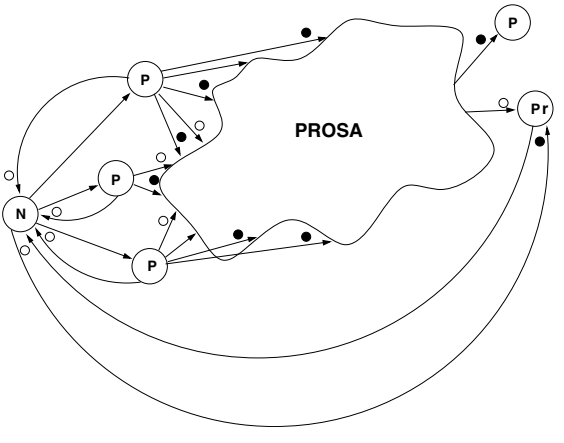


Fig. 4. Query forwarding: new FSL arises

"Philosophical Issues" and "Philosophical Perspectives", for a total amount of 750 articles.

We chose documents from these to fields in order to test if a PROSA network is able to dinamically adapt his structure, allowing "similar" peers to link together by means of Full Semantic Links or Temporary Semantic Links, and adaptively form "semantic–groups". We define a "semantic–group" as a group of semantically–linked peers that host documents belonging to a certain shared field.

Article terms have been stemmed and stored into a database. For each article, it's DV was computed, using only the most 100 frequent terms of the document. We choose to limit the number of terms of the DV because in [11] it has been proved that a larger DV does not give better results.

4.1 Simulation Results

The main target of this work is to show that a relationships–inspired network naturally evolves to a small–world. For this reason, preliminary simulations of ***PROSA*** have been focused on topological characteristics, such as clustering coefficient and average path length, because small-worlds graphs have high clustering coefficient and small average path length.

Since links netween peers in ***PROSA*** are not symmetric, it is possible to represent a ***PROSA*** network as a directed graph G(V,E). The clustering coefficient for a node in a directed graph can be defined as follows [9]:

$$CC_n = \frac{E_{n,real}}{E_{n,tot}} \tag{2}$$

where $E_{n,real}$ is the number of edges between n's neighbors and $E_{n,tot}$ is the maximum number of possible edges between n's neighbors. Note that if k is in the neighborhood of n, the viceversa is not guaranted, due to the fact that links are directed. The clustering coefficient of a graph is defined as the mean graph coefficient for all the vertices (nodes) in the graph:

$$CC = \frac{1}{|V|} \sum_{n \in V} CC_n \tag{3}$$

In figure 5 the clustering coefficient (CC) and average path length (APL) of ***PROSA*** is compared to that of an "equivalent" random graph defined as a graph with the same number of nodes and edges of the ***PROSA*** network and randomly chosen edges.

The clustering coefficient and the average path length of a random graph with $|V|$ vertices and $|E|$ edges has been computed using equations (4) and (5) [9].

$$CC_{rnd} = \frac{|E|}{|V| \cdot (|V| - 1)} \tag{4}$$

$$apl_{rnd} = \frac{\log |V|}{\log (|E|/|V|)} \tag{5}$$

These measures regard the case of ***PROSA*** networks where each peer starts with 20 documents on average. The CC and APL are computed after 10.000 queries. Each

# nodes	# edges	CC_prosa	APL_prosa	CC_rnd	ALP_rnd	CC_prosa/CC_rnd
400	15200	0.26	2.91	0.095	1.65	2.7
600	14422	0.19	2.97	0.04	2.01	4.75
800	14653	0.17	2.92	0.02	2.29	8.5
1000	14429	0.15	2.90	0.014	2.58	10.7
3000	15957	0.11	2.41	0.002	4.8	55
5000	19901	0.06	2.23	0.0008	6.17	75

Fig. 5. Clustering coefficients and APL for different network size

query contains 4 terms, on average. Looking at the results, it is clear that ***PROSA*** networks always present a higher clustering coefficient than the corresponding random graphs. This means that each peer tends to link with a strongly connected neighborhood, which represents (a part of) the "semantic group" joined by the peer. This behaviour is mainly due to the fact that links are mainly "semantic links" (both FSLs and TSLs) with nodes that provided (or requested) resources belonging to a given field. Note also that the APL for a ***PROSA*** network decreases when the number of nodes increases, while it seems to linearly depend on the network size for the correspondent random graph.

As showed in figure 6, the clustering coefficient for both ***PROSA*** and random network decreases, but the ratio between CC_prosa and CC_rnd increases almost exponentially with the number of nodes. We think this is due to the fact that in ***PROSA*** the clustering degree of the network is strictly related to the number of queries performed by nodes, expecially in this case, where the PL has a non–limited length. Figure 7 reports the ratio between number of edges and number of nodes for ***PROSA*** networks with 400 to 5000 edges (divided by 100) and the corresponding clustering coefficient.

It is clear that the clustering coefficient seems to depend on the average number of links per node. To verify this conjecture, we simulated ***PROSA*** networks behaviour for different numbers of queries, from 5000 to 20000. Results are showed in the four

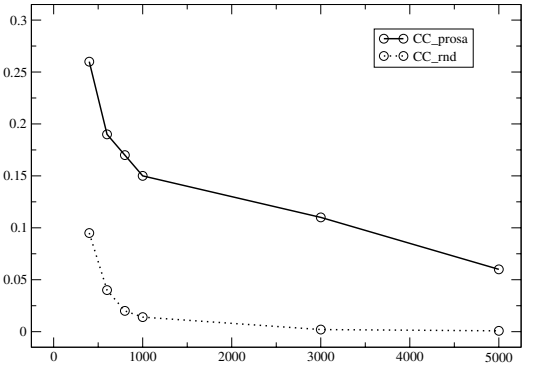


Fig. 6. Clustering coefficients for ***PROSA*** and random graph – 10.000 queries

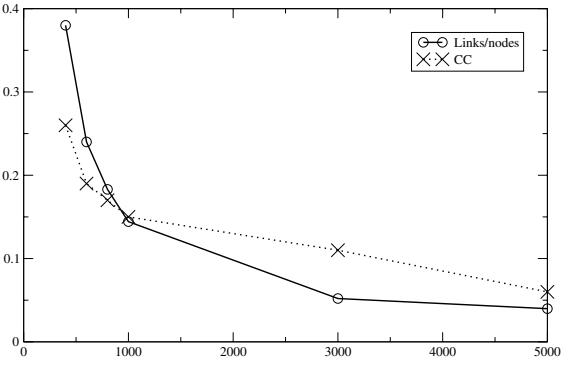
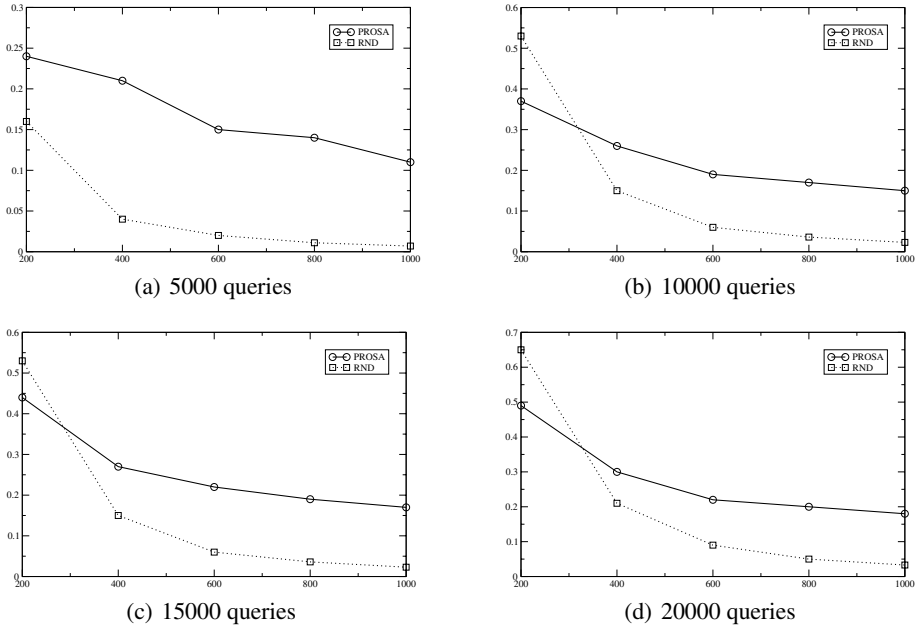


Fig. 7. Ratio between # of edges and # of nodes for different network sizes



(a) 5000 queries (b) 10000 queries

(c) 15000 queries (d) 20000 queries

Fig. 8. Clustering coefficient for ***PROSA*** and random graph for different network size and # of queries

subfigures of figure 8. These graphics show that a ***PROSA*** network has higher clustering coefficient than the corresponding random graph for networks that have more than 200 nodes.

The ratio between the clustering coefficient of ***PROSA*** networks and that of correspondent random graphs (showed in figure 9) says that ***PROSA*** clustering coefficient is always 2 to 15 times higher than that of a random graph. Due to this results, we can

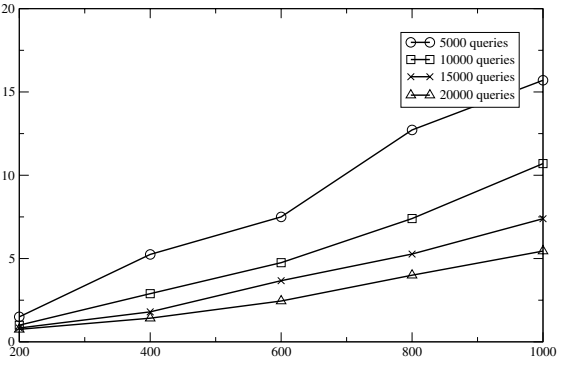


Fig. 9. Ratio between ***PROSA*** and random graph clustering coefficients, for different sizes and number of queries

finally deduce that a ***PROSA*** network evolves to a "small–world" after a number of queries which depends on the number of peers, because of the really short average path length and the relative high clustering coefficient we obtain.

5 Conclusions and Future Works

In this paper a novel P2P self–organising algorithm for resource searching and retrieving has been presented. The algorithm emulates the way social relationships among people naturally arise and evolve, and finally produces a really small–world network topology, as confirmed by simulation results. The next step is to prove that a ***PROSA*** network is internally organized into semantic–groups, i.e. highly clustered groups of peers formed by nodes that share knowledge into a certain field.

References

1. Mayank Bawa, Gurmeet Singh Manku, and Prabhakar Raghavan. Sets: search enhanced by topic segmentation. In *SIGIR '03: Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 306–313, New York, NY, USA, 2003. ACM Press.
2. V. Carchiolo, M. Malgeri, G. Mangioni, and V. Nicosia. Prosa: P2p resource organisation by social acquaintances. 2006. To be presented at AP2PC – Agents and P2P Computing Workshop at AAMAS 2 006.
3. Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46, 2001.
4. B.T. Loo, R. Huebsch, I. Stoica, and J.M. Hellerstein. The case for a hybrid p2p search infrastructure. In *Proceedings of the 3rd Internationa Workshop on Peer–to–Peer Systems (IPTPS)*, February 2004.
5. Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content addressable network. Technical Report TR-00-010, Berkeley, CA, 2000.

6. Milgram S. The small world problem. *Psychol Today*, 2:60–67, 1967.
7. Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.
8. H. Schutze and C. Silverstein. A comparison of projections for efficient document clustering. In *Prooceedings of ACM SIGIR*, pages 74–81, Philadelphia, PA, July 1997.
9. Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.
10. B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.
11. Yingwu Zhu, Xiaoyu Yang, and Yiming Hu. Making search efficient on gnutella-like p2p systems. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 56a– 56a. IEEE Computer Society, April 2005.

Towards Fully Adaptive Pipeline Parallelism for Heterogeneous Distributed Environments

Horacio González-Vélez and Murray Cole

Institute for Computing Systems Architecture
School of Informatics, University of Edinburgh, Edinburgh EH9 3JZ, UK
h.gv@ed.ac.uk, mic@inf.ed.ac.uk

Abstract. This work describes an adaptive parallel pipeline skeleton which maps pipeline stages to the best processors available in the system and clears dynamically emerging performance bottlenecks at run-time by re-mapping affected stages to other processors. It is implemented in C and MPI and evaluated on a non-dedicated heterogeneous Linux cluster. We report upon the skeleton's ability to respond to an artificially generated variation in the background load across the cluster.

1 Introduction

Pipelining is the decomposition of a repetitive sequential process into a succession of distinguishable sub-processes called *stages*, each of which can be efficiently executed on a distinct processing element or elements which operate concurrently.

In software, this approach is widely used to address grand-challenge computational science problems [1], numerical linear algebra algorithms [2], and signal processing applications [3]. Pipelines are exploited at fine-grained level in loops through compiler directives and in operating system file streams, and at coarse-grained level in parallel applications employing multiple processors. In particular, coarse-grained pipeline applications refine complex algorithms into a sequence of independent computational stages where the data is "piped" from one computational stage to another. Each stage is then allocated to a processing element in order to compose a parallel pipeline. Our pipeline follows this model.

The performance of a pipeline can be characterised in terms of *latency*, the time taken for one input to be processed by all stages, and *throughput*, the rate at which inputs can be processed when the pipeline reaches a steady state. Throughput is simply related to the processing time of the slowest stage, or *bottleneck*. When handling a large number of inputs, it is throughput rather than latency which constrains overall efficiency. Our system schedules and dynamically reschedules stages to processors, in the face of the dynamically varying processor capability which is typical of grid systems, with a view to maintaining high throughput.

The problem addressed in this paper is as follows: given a parallel pipeline program, find an effective way to improve its performance on a heterogeneous distributed environment by adapting dynamically to external load variations.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 916–926, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Our adaptive parallel pipeline has two main components: calibration and feedback. Initially the calibration is used to map stages to the best processors available in the system. Subsequently, the feedback mechanism clears performance bottlenecks at run-time by re-mapping the stages to other processors. This pipeline is implemented as a stateless skeleton in C and MPI. We present promising results of parallel executions in a non-dedicated heterogeneous Beowulf cluster, using a stage function based on a numerical benchmark.

This paper is structured as follows. First, we provide motivation for this work. Then we describe the adaptive parallel pipeline algorithm and its implementation, followed by the experimental evaluation. Finally we discuss some related approaches and make final remarks.

2 Motivation

2.1 Idealised Pipelines

We must review some generic performance issues in pipelined processing. Suppose that the original sequential process requires time t_s to process a single input. Consider an n-stage pipeline, in which t_i is the execution time for the i^{th} stage.

In an idealised model, without significant communication costs, the sequential and parallel (one processor per stage) times to process S inputs are then

$$T_{seq} = S \times t_s$$

$$T_{par} = \sum_{i=1}^{n} t_i + (S-1) \times \max(t_i)$$

$$\text{where } \max(t_i) \text{ is the bottleneck stage time}$$

It is well known that perfect pipelined performance is obtained when the stage times t_i are all equal to $\frac{t_s}{n}$, since we can then reduce the expressions to:

$$T_{seq} = S \times t_s$$

$$T_{par} = t_s + (S-1) \times \frac{t_s}{n}$$

so that as S grows large, speed-up asymptotically approaches n.

Outside this perfect situation, it is more important to reduce the bottleneck time than the latency, since the former affects the multiplicative term in T_{par}, where the former affects only the asymptotically insignificant additive term.

2.2 Pipelines on Dynamically Heterogenous Resources

With the advent of heterogeneous distributed systems, whether geographically-contiguous (clusters) or in different administrative and geographical domains (grids [4]), it is widely acknowledged that one of the major challenges in programming support is the prediction and improvement of performance [5]. Such

systems are characterised by the dynamic nature of their heterogeneity, due to shifting patterns in background load which are not under the control of the individual application programmer. The challenge is therefore to produce and support applications which can respond to this variability.

In our current work we focus on the computational aspects of heterogeneity. In what follows we make assumptions designed to keep the number of experimental variables tractable. Subsequent work will consider relaxation of these assumptions. Specifically, we assume that

- the computational weight of each stage is identical, in the sense that all stages would take the same time to process one item if executed on the same reference processor. In effect, this is to assume that the programmer has done a good abstract job of balancing stages and reducing the bottleneck. This allows us to focus on addressing issues which arise when the available processors vary dynamically in performance with respect to such a reference processor.
- communication time is not significant. Note that this is not to assume that communication is negligible, but rather to assume that communication costs hinder all stages equally.

The challenge can now be stated simply. The application programmer is required to write sequential code for the body of each pipeline stage and make a call to our pipeline skeleton to apply these stages to a set of inputs. The "grid" provides a pool of available processors. Our system maps the stages to (a subset of) the processors. It may choose to map several stages to the same processor when this processor is more powerful than the others. Periodically, our system checks the progress of the computation and may decide to remap some or all of the stages. In the following section we describe the mechanisms employed to construct this overall framework.

3 Adaptive Pipeline Parallelism

The core of our system is an algorithm for mapping pipeline stages to processors. Its main feature is that processors are calibrated at run-time, to provide the performance information upon which the mapping is based. The mapper is embedded within an iterative scheduling scheme, allowing the pipeline implementation to be adapted to prevailing conditions within the pool of available processors. We will now discuss the mapper and the rescheduler in turn.

3.1 Mapping Stages to Processors

The first step in mapping is to determine the current 'fitness' of each available processor. This is achieved by running, by way of calibration, an instance of one of the stage functions on each processor, and measuring the execution time. Any stage will do, since we have assumed that all stages are equally inherently 'heavy'. This allows us to rank processors by descending fitness (i.e. by increasing

calibration time). We can immediately discard all but the n fittest processors from the initial mapping. The mapping is generated by a greedy algorithm, which computes x_i, the number of stages to be executed by processor i.

Algorithm 1 provides a detailed description of the calibration procedure. In this algorithm, X is the aggregated array containing all x_i entries, t records the aggregated execution time of the stage function in every node, and $Chosen$ is the array of selected nodes. That is to say, P contains all nodes in the `MPI_COMM_WORLD`, $Chosen$ holds only the processors to be used, and X indicates the number of processes per node capped by the maximum processes per node. As per the greedy nature of the algorithm, every entry t_i in t takes into account the workload generated by the increasing number of stage-function instances to be executed in a given node.

We make an initial mapping in which one stage is assigned to each of the n fittest processors. We construct a ranking of the chosen processors, according to the time t_i they will take to process an item, given their currently allocated stages. Initially this is identical to the calibration ranking, but as a processor is assigned extra stages, its t_i will be the product of the number of allocated stages and its original calibration time.

The initial mapping is iteratively improved by application of the following step:

> *Consider the effect of moving one stage from the processor P_b with the highest processing time (i.e. the current bottleneck) to the processor P_l with the lowest processing time. If the new resulting processing time at P_l is smaller than the original processing time at P_b then make the switch.*

Iteration proceeds until no further improvement is possible. It is not difficult to see that such a strategy is optimal.

Suppose there was a better mapping M' than the one with which the algorithm terminates, M. Suppose that the bottleneck processor P_b in M is assigned k stages. M must assign fewer than k stages to P_b (otherwise it wouldn't better) and more stages to at least one other processor P_x (because the extra stage must be assigned somewhere). Then M can be improved by moving one stage from P_b to P_l (which may or may not be P_x, it doesn't matter), and the greedy algorithm will do so, thereby contradicting the (premature) assumption of termination. ∘

3.2 Monitoring Performance and Rescheduling

Once the pipeline is in operation, the feedback phase detects performance bottlenecks by checking whether all processors are functioning according to the initial calibration. Each stage times itself and propagates its current t_i through the pipeline, piggy backed with the real data. The final stage verifies that the T_{par} is acceptable by comparing with the original calibration times, using a fixed threshold to determine acceptability. This threshold regulates the margin before a re-calibration takes place and is expressed as a fraction of the original value.

Algorithm 1. Calibration Algorithm

```
Data: f: Stage Functions;
n: Number of Stages;
P: Nodes;
Result: Chosen: Lookup table of fittest processing elements;
X: Number of processes per Chosen node;

forall nodes in P do
    Execute f concurrently ;
    Set t_i ← execution_time(f);
end
if root node then
    set X ← 0 ;
    collect t_i into t;
    sort the P nodes ; /* Using t as key                                    */
    set ι ← 0 ;
    set ℓ ← n ;
    while ι < ℓ − 1 do
        set flag ← false ;
        set k ← ι + 1 ;
        while k < ℓ ∧ ¬flag do
            set α ← ⌊(t_k − t_i)/(t_i/x_i)⌋;
            if α ≠ 0 then
                set t_i ← t_i + t_i/x_i ;
                set x_i ← x_i + 1 ;
                set ℓ ← ℓ − 1 ;
                insert_in_order (t_i, t);
                set flag ← true ;
            end
            if ¬flag then
                ι ← ι + 1;
            end
        end
    end
    send Chosen and X to other nodes
else
    send time from this node to root node;
    receive Chosen and X;
end
```

For example, if a threshold is equal to X, it indicates that if a t_i is more than $(1 + X)$ times slower than the original worst recording, a bottleneck has been detected and a remapping is scheduled. Similarly, should the worst time be $(1 - X)$ times faster than the original best, remapping is considered.

It is crucial to note that the threshold is key to the adaptivity mechanism since a small value may cause too many remappings (thrashing) while a large one will deactivate the adaptiveness. Once a decision to remap has been made, the pipeline is allowed to drain before resuming under the new mapping.

4 Implementation

To facilitate our experiments we have designed an algorithmic skeleton [6], programmed in ANSI C with MPI to handle internode communication. Its API is

```
void pipeline(stage_t *stages, int no_stages,int in_data[], MPI_Comm comm)
```

The first parameter `stages` is an array of pointers to functions which contains the f stages, `no_stages` is n, `in_data` is the input data stream S (using simple integers here, but easily generalisable), and `comm` is a communicator encompassing the processor pool.

Based on previous experiences with skeletons on geographically dispersed grids [7], where we empirically learned the costly implications of the inherent synchronisation in collectives, we have based this design on explicit send-receive pairing. Internally, each stage is composed by a `MPI_Recv` call, the invocation to the f function, and a `MPI_Send` call.

Internally, the processor pool is represented as a lookup table of active processors. Each processor uses the table to determine its predecessor and successor. It is built during the calibration process by simply sorting the processors in the communicator by execution times. The table is also of particular importance during process migration since the migration is in essence an exchange of its entries.

On the infrastructure side, there exist a few libraries which provide MPI process migration, mainly devoted to preserving index and context variables in loops. Although they address generic MPI programs, their use requires specialised underlying distributed filesystems [8] or daemon-based services [9]. Since a key criterion to the process migration is fast process migration, we opted to develop a simple process migration mechanism based on the aforementioned process pool.

It is important to stress the fact that there are not pre-determined processors required for the execution of the pipeline. That is to say, after calibration not even the `MPI_COMM_WORLD` root process must belong to the set of fittest processes. Therefore, a re-mapping always implies a process migration with result preservation.

5 Results

The full system has been compiled with gcc 3.4.4 using "-pedantic -ansi -Wall -O2" flags and employs LAM/MPI 7.1.1. It has been deployed on a non-dedicated heterogeneous Beowulf cluster, located in the School of Informatics of the University of Edinburgh, and configured as shown in Table 1. The processors have different frequencies and exhibit different performance as determined by the BogoMips reading which is the standard Linux benchmark.

For reproducibility purposes, we have employed as stage function the `whetstones` procedure from the 1997 version [10] of the Whetstone benchmark with parameters `(256,100,0)`. It accounts for some 5 seconds of double-precision floating-point processing on an empty node in the Beowulf cluster.

Table 1. Beowulf cluster `bw240` Configuration

Nodes:	64
CPU:	Intel P4
Memory:	1 GB / node
Network:	2x100Mb/s (Shared)
OS:	Linux Red Hat FC3 - Kernel 2.6
BogoMips:	3350.52–3555.32

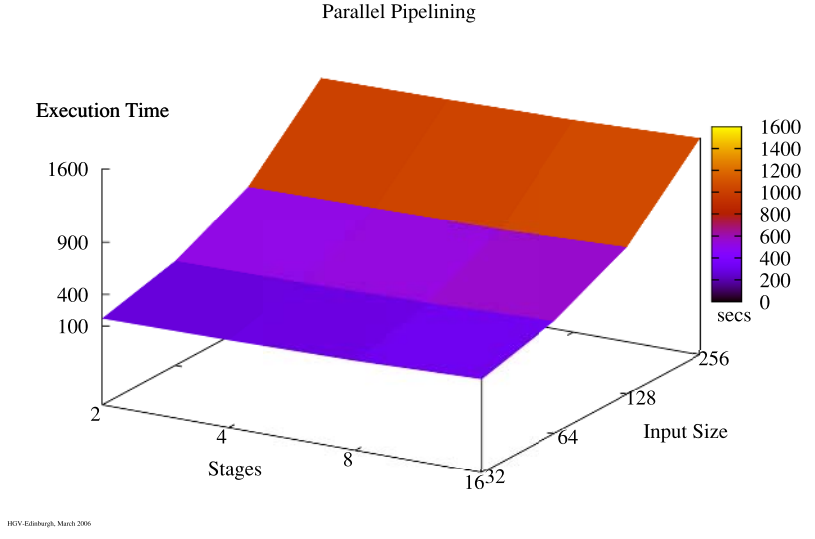


Fig. 1. Correlation between the size of the data input S and the number of stages n

Thus, all variability in the system is due to external load and, to a lesser extent, to the difference in performance among processors. All execution times reported below sum up the average of three measurements, with a standard deviation of less than 1%.

Figure 1 shows a "sanity-check" initial exploration of the parameter space, running pipelines with 2, 4, 8 and 16 identical stages, one per processor (note that a pipeline with more stages is doing more work in absolute terms) on increasingly large inputs. The execution times are primarily determined by $|S|$, the input size of the data stream, and marginally influenced by the number of stages n in the pipeline.

We have firstly explored the overhead incurred by the calibration phase. Figure 2 shows that the overhead is minimal and increases at a slow rate ($< 1\%$ for every power-of-two increment in the number of processes).

We then measured the performance impact of our system under different load conditions. Figure 3(a) depicts the times of different executions using two

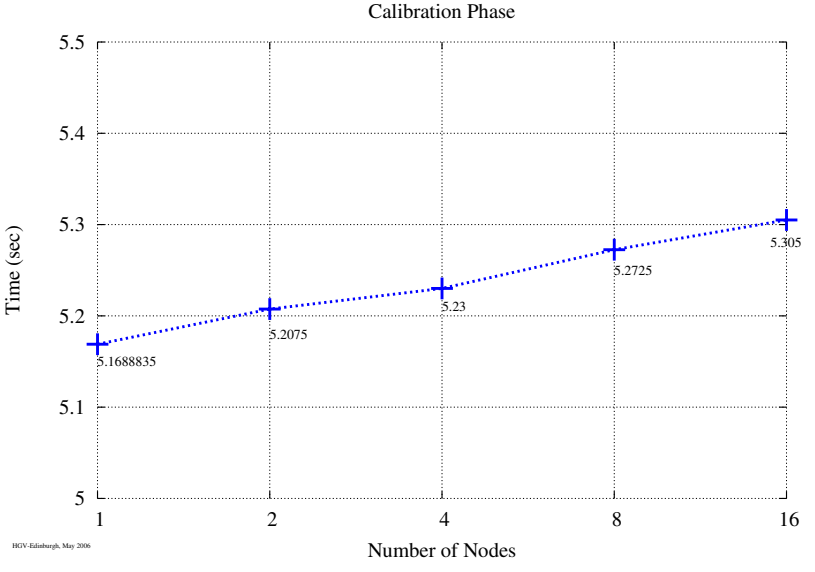


Fig. 2. Calibration phase execution times for 1,2,4,8, and 16 processes

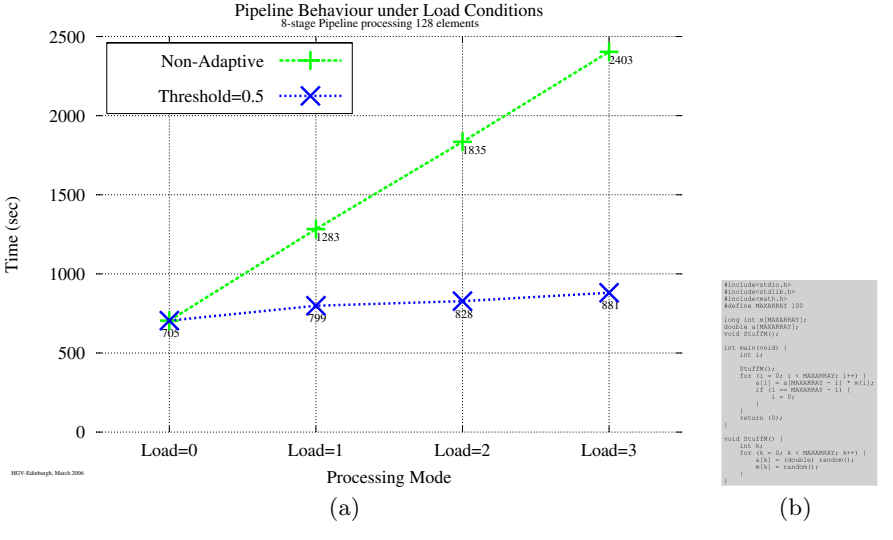


Fig. 3. (a) Comparison of the parallel pipeline using two threshold parameters: Infinite (non-adaptive) and 0.5. (b) Load generating program (adapted from [11]).

threshold parameters: infinite, which implies a non-adaptive pipeline, and 0.5. This threshold value was empirically determined using a series of test runs. Figure 3 (b) shows the actual program employed to generate load.

Taking into account the fair CPU allocation algorithm used in Linux and to assure the existence of changing load conditions, we have incrementally injected

load dynamically to the system using a simple load generation program. Each instance of this program added 1 to the load displayed by the Linux `uptime` command in a certain node ("bottleneck node") until this node became a bottleneck, while the rest of the processors did not experience any significant load variation. Thus a $Load = 0$ implies no bottlenecks, $Load = 1$ an instance of the load generator was running on the "bottleneck node" and so on. The instances were triggered after 60 seconds from the start of the program.

Figure 3(a) shows a comparison of the measured execution times with $n = 8$ and $|S| = 128$. The x-axis indicates the injected load, i.e. the number of instances of the load generator running in one processor, which were triggered during the pipeline operation.

We see that the adaptive methodology has responded well under changing load conditions, since the execution times in the non-adaptive parametrisation have increased at a considerably higher rate than the adaptive ones.

6 Related Work

The scheduling problem of the parallel pipeline construct has been previously studied in the literature.

The LLP system [12] furnishes a conceptual framework for static multi-stage allocation using algorithmic skeletons. By approaching the problem with a 0/1 knapsack problem methodology, LLP is employed to develop a theoretical solution to stage scheduling.

Based on direct-acyclic graphs, the macro-pipelining methodology [13] gives a theoretical framework for scheduling parallel pipelines. While macro-pipelining provides guidance on the coarse distribution of work to different stages, its approach is limited to dedicated digital-signal processing systems.

Another approach presents a multi-layer framework for the stage scheduling in dedicated real-time systems [14]. This work describes a series of steps to calculate end-to-end latencies based on a time-series model for a video-conferencing application. Unfortunately, it does not address the general case.

Recent work on adaptive systems [15,16] has reinforced the importance of platform adaptation for optimisation of parallel codes in heterogeneous distributed systems. While implementation of parallel pipelines can be found in several established skeletal libraries [17,18], adaptive skeletal constructs have recently started to use resource-aware mechanisms based on process algebra [19] and statistical methodologies [20], paving the way to the development of a comprehensive library of self-adaptive algorithmic skeletons for non-dedicated heterogeneous systems.

7 Conclusions

Our methodology is fundamentally different since it provides a generic system

- to pragmatically tune up the pipeline parallelism skeleton regardless of the complexity of the stage functions (calibration phase); and

– to dynamically adapt to non-dedicated heterogenous environments once the pipeline processing is established (feedback).

A close examination of the methodology will show that there is certainly room for a more instrumented approach to the determination of the re-calibration threshold. Our work provides evidence that the proposed adaptive methodology enhances pipeline parallelism performance: execution times are almost an order of magnitude greater when not using the adaptive pipeline.

It is important to emphasise this work has covered load variations attributable to different processing capabilities, while maintaining the stage function complexity constant. Although this scenario does not comprehensively address all possible pipeline applications, it certainly provides guidance on the behaviour of the general case on distributed systems.

In time, we intend to expand the experiment space by analysing pipelines with stage functions with distinct complexity and, possibly, including a pipeline-oriented application such as image processing.

In the same manner, we will study the correlation between the threshold and the stage functions. Such a study may eventually lead to the automatic determination of the optimal threshold for a given set of stages.

Acknowledgements

This work has been partly supported under the Chevening–Conacyt grant no. 81887. Horacio González–Vélez would like to thank Mary Cryan of the Laboratory for Foundations of Computer Science in Edinburgh for the early discussions on the calibration algorithm.

References

1. Pancake, C.M.: Is parallelism for you? IEEE Computat. Sci. Eng. **3** (1996) 18–37
2. Heller, D.: A survey of parallel algorithms in numerical linear algebra. SIAM Review **20** (1978) 740–777
3. Fleury, M., Downton, A.: Pipelined Processor Farms: Structured Design for Embedded Parallel Systems. Wiley-Interscience, New York (2001)
4. Foster, I., Kesselman, C., eds.: The Grid: Blueprint for a new computing infrastructure. Second edn. Morgan Kaufmann, San Francisco (2003)
5. Laforenza, D.: Grid programming: some indications where we are headed. Parallel Comput. **28** (2002) 1733–1752
6. Cole, M.: Algorithmic Skeletons: Structured Management of Parallel Computation. MIT Press, London (1989)
7. González-Vélez, H.: An adaptive skeletal task farm for grids. In: Euro-Par 2005. Number 3648 in Lect. Notes Comput. Sc., Springer-Verlag (2005) 401–410
8. Vadhiyar, S.S., Dongarra, J.: SRS: A framework for developing malleable and migratable parallel applications for distributed systems. Parallel Process. Lett. **13** (2003) 291–312
9. Sievert, O., Casanova, H.: A simple MPI process swapping architecture for iterative applications. Int. J. High Perform. Comput. Appl. **18** (2004) 341–352

10. Longbottom, R.: C/C++ Whetstone benchmark single or double precision. ftp.nosc.mil/pub/aburto (1997) Official version approved by H. J. Curnow on 21/Nov/97.
11. Gunther, N.J.: Analyzing Computer System Performance with Perl: PDQ. Springer-Verlag, Berlin (2004)
12. González, D., Almeida, F., Moreno, L.M., Rodríguez, C.: Towards the automatic optimal mapping of pipeline algorithms. Parallel Comput. **29** (2003) 241–254
13. Banerjee, S., Hamada, T., Chau, P.M., Fellman, R.D.: Macro pipelining based scheduling on high performance heterogeneous multiprocessor systems. IEEE Trans. Acoust. Speech Signal Process. **43** (1995) 1468–1484
14. Chatterjee, S., Strosnider, J.K.: Distributed Pipeline Scheduling: A framework for distributed, heterogeneous real-time system design. Comput. J. **38** (1995) 271–285
15. Vadhiyar, S.S., Dongarra, J.J.: Self adaptivity in grid computing. Concurrency Computat. Pract. Exper. **17** (2005) 235–257
16. Kelly, P.H.J., Beckmann, O.: Generative and adaptive methods in performance programming. Parallel Process. Lett. **15** (2005) 239–255
17. Aldinucci, M., Danelutto, M., Teti, P.: An advanced environment supporting structured parallel programming in Java. Future Gener. Comput. Syst. **19** (2003) 611–626
18. Cole, M.: Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming. Parallel Comput. **30** (2004) 389–406
19. Yaikhom, G., Cole, M., Gilmore, S.: Combining measurement and stochastic modelling to enhance scheduling decisions for a parallel mean value analysis algorithm. In: ICCS 2006. Number 3992 in Lect. Notes Comput. Sc., Springer-Verlag (2006) 929–936
20. González-Vélez, H.: Self-adaptive skeletal task farm for computational grids. Parallel Comput. (2006) In press doi:10.1016/j.parco.2006.07.002.

Compiler-Directed Energy-Time Tradeoff in MPI Programs on DVS-Enabled Parallel Systems*

Huizhan Yi, Juan Chen, and Xunjun Yang

Section 620, School of Computer,
National University of Defense Technology,
Changsha, 410073, Hunan, P.R. China
{huizhanyi, juanchen, xjyang}@nudt.edu.cn

Abstract. Although parallel systems with high peak performance have been exciting, high peak performance often means high power consumption. In this paper, power-aware parallel systems are investigated, where each node can make dynamic voltage scaling (DVS). Based on the characteristics of communication and memory access in MPI programs, a compiler is used to automatically form communication and computation regions, and to optimally assign frequency and voltage to the regions. Frequency and voltage of each node are dynamically adjusted, and energy consumption is minimized within the limit of performance loss. The results from simulations and experiments show that compiler-directed energy-time tradeoff can save 20~40% energy consumption with less than 5% performance loss.

1 Introduction

Manufacturers in high-performance computing have been pursuing parallel systems with high peak performance. Parallel systems with high peak performance often bring about large power consumption. Earth simulator, for instance, has 18MW peak power consumption [1], and another parallel system, Blue Gene/L, has 1.6MW peak power consumption [2]. Large energy consumption has increased the operating cost. Furthermore, high power consumption has led to the increase of environment temperature, and as a result, system reliability has been reduced. Hsu, et al presented some data on the reliability of high-performance systems, and the results showed that the reliability was significantly reduced when environment temperature is increased [3]. At the same time, limited by network and memory performance, the systems with high peak performance often cannot obtain high real performance, and they have lower energy efficiency.

Power consumption from computation nodes is one of the largest power components in parallel systems. Parallel execution of parallel applications often does not completely utilize the ability of computation nodes. Limited by network and memory performance, computation nodes often could be idle. Although they keep

* Supported by the National High Technology Development 863 Program of China under Grant No. 2004AA1Z2210 and Server OS Kernel under Grant No. 2002AA1Z2101.

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 927 – 938, 2006.
© Springer-Verlag Berlin Heidelberg 2006

idle, computation nodes always run in the highest speed and consume large energy consumption. Therefore, when computation nodes become idle, we can adjust their frequency/voltage [4] and reduce the running speed, and the performance loss, however, is minimized. That is to say, with some minor performance loss, we can get large energy savings. In the past, dynamic voltage scaling (DVS) is often used in embedded systems to improve energy efficiency of a single processor. Recently, some attempts of DVS are to establish parallel clusters with DVS-enabled processors. Ge, et al have established a cluster with 16 nodes, in which each node is a Pentium M processor [5]. Freeh, et al have given a 10-node cluster, and each node is an Athlon-64 processor [6]. The results show that DVS-enabled parallel systems can obtain large energy savings with some minor performance loss.

The past work has been using manual methods to explore energy-time tradeoff, and does not propose an automatic method to perform it. As a result, it is difficult of them to make tradeoff for real operational applications. In this paper, we present compiler-directed energy-time tradeoff in MPI programs on DVS-enabled parallel systems. Parallel applications often consist of computation and communication at the same time. In computation regions, each node performs parallel computation; in communication regions, computation nodes exchange data each other. Currently, interconnection networks have lower performance than that of computation nodes, and the communication is often the performance bottleneck. Another bottleneck is the memory. Network and memory often make computation nodes idle. So we divide parallel applications into computation regions and communication regions, and instrument parallel applications. By real execution, we obtain the execution time of each region on different frequency levels. Based on the specified limit of performance loss, we formulate it as a 0-1 integer-programming problem (IP), and obtain the optimal frequency/voltage assignment for the regions, which minimizes the energy consumption.

The rest of this paper is organized as follows. In Section 2, we review the related work. In Section 3 we interpret the simulation environment in detail, and analyze some typical communication and computation applications. In Section 4, we give the compiler-directed energy-time tradeoff technique. In Section 5, we present the simulation and experiment results. In the last section, we conclude the paper.

2 The Related Work

Initially, dynamic voltage scaling is used in desktop and embedded systems. Weiser, et al presented the earliest OS-directed voltage scheduling [7]. Lorch summarized the low-power techniques on OS level, including the DVS [8]. Mosse, et al proposed compiler-directed real-time dynamic voltage scheduling [9]. Hsu, et al analyzed compiler-directed energy-time tradeoff in a single processor [10].

Recently, some work investigated dynamic voltage scaling in high-performance computing domain, and the typical work is the tradeoff between energy and time. Ge, et al analyzed the feasibility of energy-time tradeoff in DVS-enabled parallel system, and proposed three kinds of DVS scheduling methods [5]. Freeh, et al gave some detailed cases of MPI applications to prove that large energy savings is possible with

a few performance loss [6]. Hsu, et al presented a parallel voltage scheduling technique on OS level [3]. Springer, et al proposed the problem how to minimize the execution time when the energy consumption is within the user-defined limit [11]. The common characteristic of the work is to perform the energy-time tradeoff according to the user's experience, and it is difficult for them to be used in real operational applications.

3 Simulation Environment

3.1 *MIPSpar*: A DVS-Enabled Parallel System

We establish a cycle-accurate performance/power simulation environment named *MIPSpar*. The performance simulation of *MIPSpar* has used the parallel simulator, ISIS [12]. Each node of ISIS cycle-accurately simulates MIPS instruction set, model in-order pipelining structure, and consists of the divided instruction cache and data cache. Each node models the detailed bus and memory access. By a network interface (NI), each node connects to the interconnect network. ISIS can model distributed and shared memory architectures, and we only use the distributed memory architecture. In the parallel systems, each node has its local memory, the interconnection network connects the nodes, and using the data in another node need explicitly pass message between them. We have use the 2D-torus direct network, as shown in Figure 1. Each node includes a router, and the network links connect the routers. Each router uses the wormhole switching technique, and uses the virtual channel technique to avoid deadlock. We extend the interconnection network, and add the credit mechanism that supports parametric network latency and bandwidth.

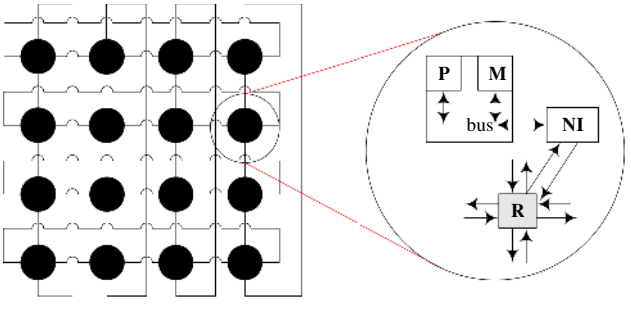


Fig. 1. The architecture of the parallel system

We add the support for dynamic voltage scaling in ISIS, and each node can separately adjust frequency and voltage. We use the frequency configuration of Athon64 3200+ [3], as listed in Table 1. We suppose that the power consumption of the node is 50W when the frequency is 2GHz, and as a result, the effective switching capacitance is 11.11nF. Use the formula $P=C_{eff} f V^2$, we have 7W power consumption when the frequency is 800MHz. The frequency and voltage adjustment has time and energy

overhead. A detailed statistics from Hsu's [13] shows that the switching time often needs a few microseconds, and we set 10 μs time overhead. The energy overhead is computed by the formula 1 [14]

$$E = (1-\eta) \cdot C \cdot | V_2^2 - V_1^2 | \quad (1)$$

where η is the switching efficiency, and generally is set to 90%. C is switching capacitance with 11.11nF, and V_1 and V_2 are the voltages at the beginning and end of the switching.

Table 1. The voltage and frequency configuration

f(GHz)	*V*	*f*(GHz)	*V*
0.8	0.9	1.6	1.3
1.0	1.0	1.8	1.4
1.2	1.1	2.0	1.5
1.4	1.2		

The performance simulator only can give the execution cycles, and in order to adjust frequency, we add a parameter to accumulate the execution time. In the simulation environment, the nodes can make dynamic voltage scaling, and the memory and network is run in fixed frequency. Therefore, after the frequency adjustment, the relative performance of the three components is changed. We divide the whole system into three clock domains, and exhibit the performance variation by changing the memory and network latency. Currently, the latency of memory access is 10^1~$10^2 ns$ [15], the network latency is 10^{-1}~10^2 μs [16]. Therefore, when the frequency is 2GHz, we define the memory latency as 50 cycles, and define network latency as 200 cycles. For other frequencies, the memory latency and network latency are computed by

$$lat_m = f / 40$$

and

$$lat_n = f / 10$$

The simulation environment consists of a MPI library, Osiris. Osiris library includes basic blocked point-to-point MPI communication functions. Based on the point-to-point communication functions, we add some collective communication functions [17]. In order to support users or compilers to adjust voltage and frequency, we add a frequency adjustment function:

MPI_Setfreq(int *freq*)

Users or compilers can modify the system frequency by calling the function. The function changes the system frequency by writing a fixed device port. According to the frequency configuration, the system adjusts the voltage and frequency to the corresponding level at the same time. Therefore, when we refer to the frequency adjustment, we always adjust the voltage and frequency simultaneously.

3.2 The Effect of DVS for Communication and Computation

We investigate the effect of DVS for communication and computation on *MIPSpar*. We select the representative communication functions, which consist of point-to-point communication, broadcast, reduce, allreduce, barrier, and alltoall. We use five applications from StreamIt's [18] to investigate the effect of DVS for computation, which include bitonic, fft, firref, matmul, and matrix.

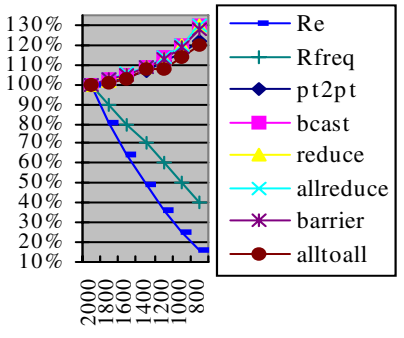


Fig. 2. The normalized execution time on 16-node systems for communication functions

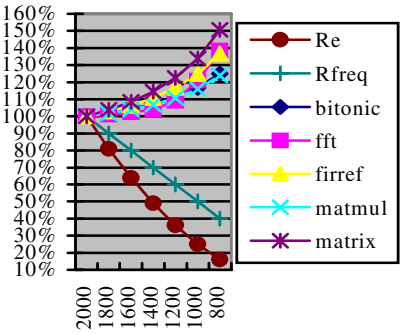


Fig. 3. The normalized execution time for computation functions

We investigate the effect of DVS for communication functions on a 16-node system, as shown in Figure 2. Here, the x-axis is the clock frequency, and y-axis represents the ratios. R_e indicates the ratios of energy consumption of current frequency versus maximum frequency, and R_{freq} indicates the ratios of current frequency versus maximum frequency. The reduced curves present the ratios of execution time of current frequency versus maximum frequency. The results show that the decreased frequencies have resulted in the minor performance loss of communication functions, and at the same time, we obtain large energy savings. Due to the limit of network performance, there are large quantities of idle cycles, and as a result, the decreased frequency results in large energy savings with some minor performance loss. Furthermore, there is the same characteristic for every communication function, and there is no obvious distinction for the communication functions.

As shown in Figure 3, we illustrate the effect of DVS for computation functions. Compared with communication functions, there is much significant effect for computation functions. The execution time of matrix, for instance, has increased 50% when the frequency is reduced to 800MHz. Moreover, there are different effects for different functions. For example, the ratio of the prolonged execution time of matrix is larger than that of bitonic by 20%.

4 Compiler-Directed Energy-Time Tradeoff on DVS-Enabled Systems

Based on the analyses of application characteristics, we see that DVS can significantly reduce energy consumption of parallel applications with some minor performance loss.

Therefore, if there is no special performance need from user, we can run applications with lower frequency and get larger energy savings. Furthermore, there are different characteristics between communication regions and computation regions. We divide an applications into communication regions and computation regions, and within the user-defined limit of performance loss, we assign the optimal frequency for different regions to minimize energy consumption. Specifically, we integrate the communication functions and their neighboring computation into some communication regions, and the reduced part of the application is formed into a computation region. By instrumentation and execution, we obtain the performance loss of different regions, and finally we assign the optimal frequency for different regions and minimize the energy consumption by solving a 0-1 integer-programming problem.

Next, we present the problem in detail. Assume that an application consists of n regions R_i, i=0,…, n-1. Here, R_i, i=1,…, n-1 are communications regions, and R_0 is the reduced part, that is to say, a computation region. Suppose that the execution time of R_i on frequency f is defined as $T(R_i, f)$, and the power consumption is denoted as P_f. The time overhead of DVS is T_{tr}, the energy overhead of DVS is P_{tr}, and the times of frequency switching are N_{tr}. The performance loss does not exceed the p percent of the execution time on maximum frequency. If the program is run in m nodes, we compute the optimal frequency configuration for each node. We have m 0-1 integer programming problem:

$$MIN_{\delta}\left(\sum_{i,f}\delta(R_i^j,f)\cdot P_f\cdot T(R_i^j,f)+P_{tr}\cdot N_{tr}^j\right)$$

subject to

$$\sum_{i,f}\delta(R_i^j,f)\cdot T(R_i^j,f)+T_{tr}\cdot N_{tr}^j\le(1+p)\cdot T(P^j,f_{\max})$$

where j =1,…,m-1. Here, P^j is the whole application on the jth node. $\delta(R_i^j,f)$ is a variable defined as 0 or 1. $\delta=1$ means the ith region on the jth node is executed in the frequency f. Since each region has used only one frequency, we have

$$\sum_f\delta(R_i^j,f)=1,\ i=1,\ldots,n\text{-}1,\ j=1,\ldots,m\text{-}1.$$

Let $N(R_0,R_i)$ denote the transition number between R_0 and R_i, then the times of frequency switching are

$$N_{tr}^j=\sum_i\left(N(R_0^j,R_i^j)\cdot\sum_f|\delta(R_0^j,f)-\delta(R_i^j,f)|\right)$$

Integer programming is a NP-complete problem, and need a heuristic algorithm. Since we have not formed too many regions, we just enumerate all the cases.

Here, we need form communication regions and computation regions, and then we need the execution time $T(R_i, f)$ and the transition number $N(R_0,R_i)$. Since the frequency adjustment has time overhead, the frequency switching between the regions with too short execution time often cannot obtain energy savings. We analyze the programs by compiler, and if two communication regions are spaced by a computation region with too short execution time, we combine two communication regions and the computation region. By instrumentation and execution, we obtain $T(R_i, f)$ and

$N(R_0,R_i)$.Then, we solve the 0-1 integer programming problem and get the optimal frequency configuration. Finally, we insert the frequency adjustment functions into the boundary of the regions. The related techniques are as follows.

4.1 Select the Region Size

The basic communication region is formed by a single MPI communication function. Next, we analyze the condition that two regions can be combined. Suppose two regions are spaced by a computation region. If we take two communication regions as independent regions, then the execution time of the middle computation region is $T+2*T_{tr}$, where T is the execution time of the computation region on maximum frequency, and T_{tr} is the time overhead of voltage adjustment. Or else, we combine two communication regions and the middle computation region. If we assign x percent of the maximum frequency to the combined region within the limit of performance loss and the combined region uses a single frequency, then the longest execution time of the original computation region is T/x. In order to obtain energy savings, we have the inequality $T+2*T_{tr} \le T/x$. That is to say, $T \ge 2*T_{tr}/(1/x-1)$. Here, we have the minimum execution time of the middle computation region.

Next, we need give an estimation of x. Here, we use the average method. If we confine the performance loss to no more than p percent of the execution time on maximum frequency, the average frequency of the whole program can be set to $1/(1+p)$ of maximum frequency. We approximate x with $1/(1+p)$, and therefore, we have $T \ge 2*T_{tr}/p$.

Generally speaking, the minimum frequency should be less than $1/(1+p)$ of maximum frequency. Our approximation can lead to combine more communication regions, and obtain much larger communication regions. The combination assists in reducing the effect of voltage adjustment overhead, and decreases the region number.

4.2 Estimate the Execution Time of the Computation Intervals

Combining the communication regions need estimate the execution time of the computation region. In high performance parallel system, the hardware features and application characteristics make it impossible to statically analyze the execution time by compiler, and the time estimation must integrate profiling with compiler techniques. Fortunately, although we need the execution time, the accurate time estimation is not necessary. We just need a time estimation technique that has moderate precision. Saavedra-Barrera proposed a time estimation method based on the time estimation of FORTRAN source language constructs [19]. A FORTRAN construct set was summarized that consisted of arithmetic operators with local/global operands, conditional and logical operators, execution control and array access operators, and intrinsic functions. The execution time of the basic language constructs was tested directly or indirectly on the destination system. From the execution time of the basic constructs and application characteristics, the running time of a given benchmark on a give machine was predicted with good accuracy.

We use the time estimation technique from Saavedra-Barrera. We give a C language construct set and use the similar classification. In order to estimate the execution time

of an application, we need to know the exact number of loop iterations and the *taken* ratio of condition branch. We obtain the information by profiling techniques. In GCC compiler, there is a test coverage tool GCOV. GCOV can give the execution number of the program source code line. With some command line parameter related to GCOV, GCC compiler compiles the MPI applications. The executable program is executed in final parameters, and outputs some execution profiles. After GCOV processes the outputs, we get the execution number of source code line and the *taken* ratio of condition branch. Using the information, we finally get the average iteration number of each loop and the *taken* ratio of each condition branch. Integrating the language and application characteristics, we can obtain the time estimation by compiler. For our applications, the time estimation precision is between 50~100%, which is lower than that of the original paper (less than 20%). Despite the lower precision, it is enough for our demand. Moreover, the estimation technique is fast, and is proper to estimate the time of large parallel applications.

4.3 Form the Communication Regions

We form the communication regions on syntax tree of parallel applications. Using the time estimation technique, we give the time of computation region. If the execution time is smaller than $2*T_{tr}/p$, we combine two communication regions into one region.

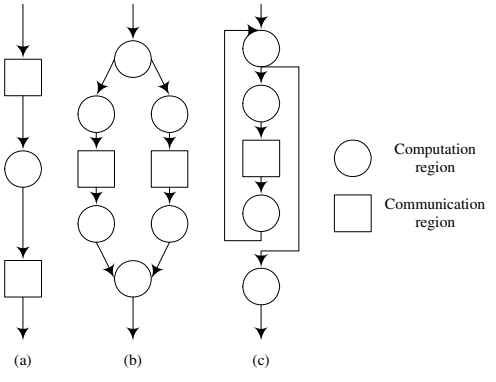


Fig. 4. The program structures. (a) Sequence; (b) Condition; (c) Loop.

By scanning the syntax tree, we search for all the communication function calls, and form the initial communication regions. We analyze the neighboring program structure of each communication region, estimate the corresponding computation time, and judge whether they can be combined into one region. Now we analyze three program structure, as shown in Figure 4. Here, the circles represent the computation regions that do not include communication directly or indirectly, and the squares are the communication regions. For the sequence of two communication regions, we estimate the middle computation region, and combine the two communication regions if the execution time of the computation region is less than $2*T_{tr}/p$. For the condition

structure, if the execution time of the computation interval in the condition structure is less than $2*T_{tr}/p$, we form the condition structure into one region. For the communication region included in a loop, we estimate the computation time of the whole loop, and if the time is less than $2*T_{tr}/p$, we let the loop as a single region. We do not stop the process until we cannot find out any region that can be combined. From the analyses, we can see that a communication region is a code segment that has only one entrance and one exit. We tag each region with a unique number, and insert instrumentation codes at the beginning and end of the communication region, which accumulate the execution time and the transition number.

4.4 Compiler-Directed Energy-Time Tradeoff

We give the course of compiler-directed energy-time tradeoff. Using the GCC/GCOV tools, we compile and run MPI applications, and get the profiles on the iteration number of loops and the taken probability of branch structures. Using the profile information and the database of the execution time of high-level language structure, we form the communication regions on syntax tree and instrument the applications. Then, we execute the instrumented applications at each frequency, and get the execution time and transition number of each region. According to the information, we solve the integer-programming problem within the limit of performance loss, and get the optimal frequency assignment for each region. Finally, the compiler sets the optimal frequencies, and output the final object files.

5 Experiments

Parallel simulation for large applications is often impossible since the simulation time is too long. We select four small MPI parallel applications. Three of them come from the Osiris library, which are FFT, BITONIC, and raytrace(RT). FFT is a MPI version of Fast Fourier Transform, BITONIC is a MPI version of fast bitonic sort, and raytrace is a MPI version of ray trace. The last application is IS, which comes from NPB suite. The final region number of four applications is listed in Table 2. Since the communication often give more chance to reduce frequency, we further investigate the ratio of the execution time of the communication functions versus the whole applications in NPB suite in a cluster.

Table 2. The number of the regions

App	No. of regions	App	No. of regions
FFT	5	RT	3
BITONIC	4	IS	5

5.1 The Energy-Time Tradeoff

We set 5% performance loss, and present the energy and time variation on 16 nodes in Figure 5. Here, FFT, BITONIC, and raytrace have used three input data set: 1024,

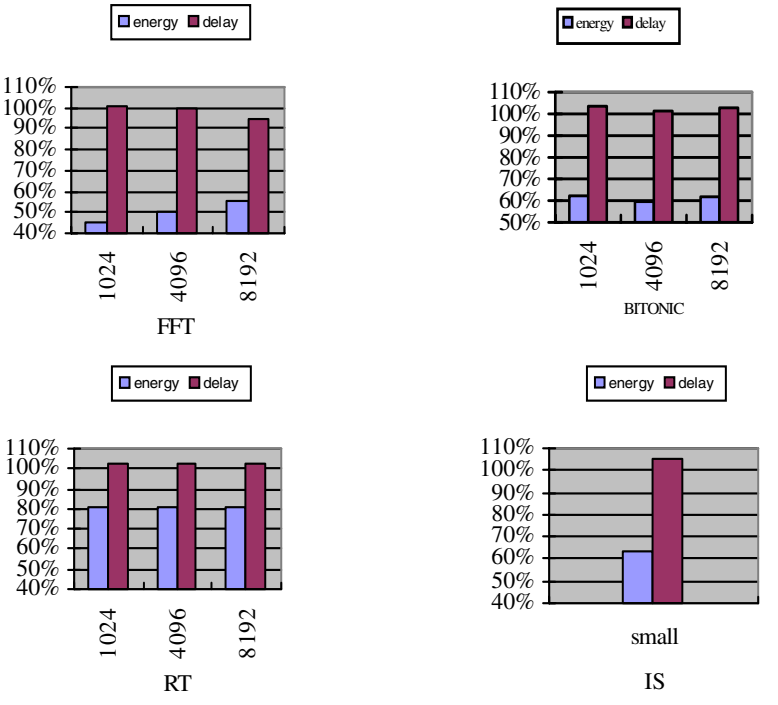


Fig. 5. The energy and time variation within the limit of 5% performance loss

4096, and 8192, and IS uses the small data set. As shown in Figure 5, with some minor performance loss, all applications obtain large energy savings. The savings of FFT, BITONIC, and IS have exceeded 40%. Raytrace also gets 20% energy savings. Moreover, the performance loss is confined in user-defined limit. For the FFT application, some minor performance improvement is observed, which means that the imbalance of computation node versus memory and network performance deteriorates system efficiency.

5.2 The Ratio of Execution Time of Communication Functions

We investigated the ratio of execution time of communication functions in NPB suite. The applications are executed in a cluster that includes 32-node SMP system. Each node consists of two P4 Xeon 3.4 GHz processors, and all the nodes are connected in G-bit Ethernet. The number of the supported maximum process is 64. NPB suite is often used to investigate the performance of MPI library of parallel systems, includes eight applications (BT, CG, EP, FT, IS, LU, MG, SP), and each application has four input data sets (S,A,B,C), which represent different sizes. The investigated object regions mainly include communication functions, and combine some communication functions spaced by small computation regions sometimes.

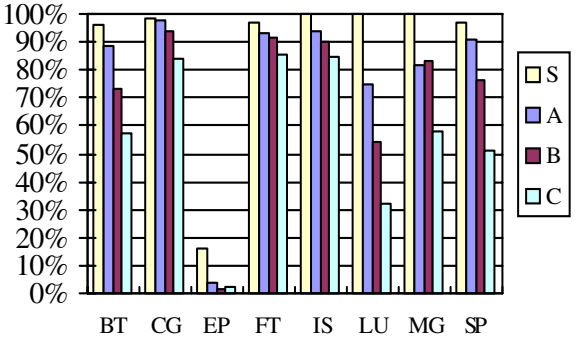


Fig. 6. The ratio of the execution time of the communication functions on 16 nodes

We present the ratio of the execution time of communication functions versus the whole applications on 16 nodes, as shown in Figure 6. Many applications have the high ratio of communication, and so it is proper to define the communication as an independent region. With the enlargement of the input data set, the ratio of communication will decrease, and the ratio of computation will increase.

6 Conclusions

In this paper we investigate compiler-directed energy-time tradeoff. Based on the analyses of computation and communication characteristics, we divide the applications into computation and communication regions by compiler, and investigate the effect of frequency adjustment by instrumentation and execution. We assign the optimal frequency to different regions, and obtain the minimum energy consumption within the limit of performance loss. The results from simulations and experiments show the techniques can effectively make energy-time tradeoff.

References

1. X. Feng, R. Ge, and K. W. Cameron , Power and Energy Profiling of Scientific Applications on Distributed Systems, 19th International Parallel and Distributed Processing Symposium (IPDPS 05), April 2005. (Denver, CO).
2. The BlueGene/L Team. An Overview of the BlueGene/L Supercomputer. SC'2002, November 16-22, 2002, Baltimore, USA.
3. Chunghsing Hsu and Wuchun Feng, A Power-Aware Run-Time System for High-Performance Computing, SC'05 November 12-18, 2005, Seattle, Washington, USA.
4. Trevor Pering, Tom Burd, and Robert Brodersen. Dynamic Voltage Scaling and the Design of a Low-Power Microprocessor System. In Proc. Power-Driven Microarchitecture Workshop, associated with ISCA98. Barcelona, Spain, June 1998.
5. Rong Ge, Xizhou Feng, Kirk W. Cameron, Performance-constrained Distributed DVS Scheduling for Scientific Applications on Power-aware Clusters, SC'05 November 12-18, 2005, Seattle, Washington, USA.

6. Vincent W. Freeh, Feng Pan, Nandini Kappiah, David K. Lowenthal, Rob Springer, Exploring the Energy-Time Tradeoff in MPI Programs on a Power-Scalable Cluster, 19th International Parallel and Distributed Processing Symposium (IPDPS 05), April 2005. (Denver, CO).
7. Weiser, W., Welch, B., Demers, A., and Shenker, S. Scheduling for Reduced CPU Energy. Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation, November 1994, pp. 13-23.
8. Jacob Rubin Lorch. Operating Systems Techniques for Reducing Processor Energy Consumption [Ph.D. thesis]. UNIVERSITY of CALIFORNIA, BERKELEY, Fall 2001.
9. Daniel Mosse, etc. Compiler-Assisted Dynamic Power-Aware Scheduling for Real-Time Applications. Workshop on Compilers and Operating Systems for Low-Power (COLP'00), Philadelphia, PA, October 2000.
10. Chung-Hsing Hsu, etc. The Design, Implementation, and Evaluation of a Compiler Algorithm for CPU Energy Reduction. Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation, pp. 38--48, June 2003.
11. Robert Springer, David K. Lowenthal, Barry Rountree, Vincent W. Freeh, Minimizing Execution Time in MPI Programs on an Energy-Constrained, Power-Scalable Cluster, PPoPP'06 March 29–31, 2006, New York, New York, USA.
12. Masaki Wakabayashi, Hideharu Amano, "Envionment for Multiprocessor Simulator Development", Fifth International Symposium on Parallel Architectures, Algorithms, and Networks, 2000, pp.64-71, Dec 2000.
13. Chung-Hsing Hsu, COMPILER-DIRECTED DYNAMIC VOLTAGE AND FREQUENCY SCALING FOR CPU POWER AND ENERGY REDUCTION[Ph.D, thesis], The State University of New Jersey, October, 2003.
14. Thomas D. Burd, Robert W. Brodersen, Design Issues for Dynamic Voltage Scaling, ISPLED'00, Rapallo, Italy, 2000.
15. RAMESH RADHAKRISHNAN, JIM PANKRATZ, Introducing DDR2 Memory in Eighth-Generation Dell PowerEdge Servers for Improved Performance, Dell Power Solutions, October 2004.
16. Voltaire Inc., InfiniBand: The Next Step in High Performance Computing, A Voltaire White Paper, February 2003.
17. R. Thakur, R. Rabenseifnerm and Q.Gropp, Optimization of Collective Communication Operations in MPICH, In International Journal of High Performance Computing Applications, 2005.
18. William Thies, StreamIt: A Compiler Infrastructure for Stream Programs, In International Conference on Compiler Construction, Grenoble, France, Apr. 2002.
19. Rafael H. Saavedra-Barrera, "Machine Characterization and Benchmark Performance Prediction", Berkeley, California, June 30, 1988.

A GPGPU Approach for Accelerating 2-D/3-D Rigid Registration of Medical Images

Fumihiko Ino[1], Jun Gomita[2], Yasuhiro Kawasaki[1], and Kenichi Hagihara[1]

[1] Graduate School of Information Science and Technology, Osaka University
1-3 Machikaneyama, Toyonaka, Osaka 560-8531, Japan
ino@ist.osaka-u.ac.jp
[2] Graduate School of Information Science and Technology, The University of Tokyo

Abstract. This paper presents a fast 2-D/3-D rigid registration method using a GPGPU approach, which stands for general-purpose computation on the graphics processing unit (GPU). Our method is based on an intensity-based registration algorithm using biplane images. To accelerate this algorithm, we execute three key procedures of 2-D/3-D registration on the GPU: digitally reconstructed radiograph (DRR) generation, gradient image generation, and normalized cross correlation (NCC) computation. We investigate the usability of our method in terms of registration time and robustness. The experimental results show that our GPU-based method successfully completes a registration task in about 10 seconds, demonstrating shorter registration time than a previous method based on a cluster computing approach.

1 Introduction

Image registration technique [1,2] plays an increasingly important role in computer-aided surgery. For example, as illustrated in Fig. 1, 2-D/3-D registration technique allows us to align a preoperative 3-D CT volume with an intraoperative 2-D fluoroscopy image, giving us point correspondences between the coordinates in the virtual world and those in the real world. These precise correspondences are necessary to exactly perform a preoperative plan in the real world, which is carefully developed using the preoperative volume in advance of surgery. However, naive CPU implementations take several minutes to complete a registration task due to a large amount of computation. Therefore, some acceleration techniques are required to use this technique for surgical assistances, where response time is strictly limited in a short time.

One emerging computational platform is the graphics processing unit (GPU), namely commodity graphics hardware, which is rapidly increasing performance beyond Moore's law [3]. For example, nVIDIA's GeForce 6800 provides approximately 120 GFLOPS at peak performance, which equals to six 5-GHz Pentium 4 processors [4]. Furthermore, recent GPUs provide programmability to users, making themselves a more flexible platform as compared with earlier non-programmable GPUs, which deal only with rendering tasks of 3-D objects. Therefore, many researchers are trying to apply the GPU to a variety of problems such as a fluid dynamics simulator [5], numerical application [6], data clustering application [7], and so on.

The objective of our work is to achieve fast 2-D/3-D registration by means of a general-purpose computation on the GPU (GPGPU) approach [8]. We implement the

M. Guo et al. (Eds.): ISPA 2006, LNCS 4330, pp. 939–950, 2006.
© Springer-Verlag Berlin Heidelberg 2006

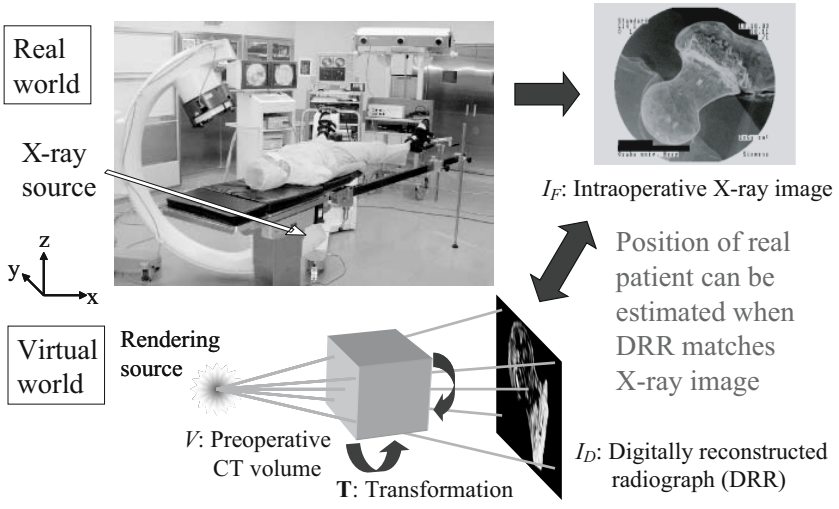


Fig. 1. Overview of 2-D/3-D registration

key procedures of a registration algorithm [9] on the GPU: 1) digitally reconstructed radiograph (DRR) generation; 2) gradient image generation; and 3) normalized cross correlation (NCC) computation. The main contribution of our work is the GPU implementation for procedures 2) and 3) based on that for procedure 1) [10]. We compare our GPU-based method with a cluster-based method [9] in terms of performance and robustness. Our method differs from prior methods [10,11], which employ different strategies to implement a part of the three procedures on the GPU.

The rest of the paper is organized as follows. We begin in Section 2 by introducing the 2-D/3-D registration algorithm, and then show an overview of GPU architecture in Section 3. We then present our GPU-based method in Section 4. Section 5 shows some experimental results. Finally, Section 6 concludes the paper.

2 2-D/3-D Registration Algorithm

The problem of 2-D/3-D registration is to compute the rigid transformation parameter **T** that relates the coordinate system of a 3-D volume V and that of a 2-D image I_F (usually, a fluoroscopy image).

We first describe a single-image version of the registration algorithm for easier understanding of the biplane-image version [12]. Our method is based on an intensity-based algorithm [1,13], which resolves the registration problem into an optimization problem. The algorithm optimizes a cost function C associated with transformation parameter **T**, where **T** represents the translation and rotation of V. The cost function C here represents the similarity between an image I_F and a DRR I_D produced by projection of V. The optimization is done by the steepest descent optimization technique [14] in a coarse-to-fine manner.

According to an empirical study [13], we currently use gradient correlation (GC) for the cost function C. Given two 2-D images, A and B, GC $G(A, B)$ between them is given by:

$$G(A,B) = \frac{1}{2}\left[N\left(\frac{\partial A}{\partial x}, \frac{\partial B}{\partial x}\right) + N\left(\frac{\partial A}{\partial y}, \frac{\partial B}{\partial y}\right)\right] \tag{1}$$

where N represents NCC between two images, $\partial A/\partial x$ and $\partial A/\partial y$ ($\partial B/\partial x$ and $\partial B/\partial y$) are the horizontal and the vertical gradient images of A (B, respectively). NCC $N(A, B)$ between $n \times n$ pixel images A and B is given by:

$$N(A,B) = \frac{S_{AB} - S_A S_B/n^2}{\sqrt{S_{A^2} - (S_A)^2/n^2}\sqrt{S_{B^2} - (S_B)^2/n^2}}, \tag{2}$$

where S_A and S_{A^2} (S_B and S_{B^2}) represent the sum and the squared sum of A (B), respectively, and S_{AB} represents the multiplied sum of A and B.

The gradient images are produced by the first derivative of a Gaussian. This filter enhances the outline of objects with reducing and smoothing noise in images. Therefore, it contributes to improve the robustness of registration. Given an image A, the horizontal gradient image $\partial A/\partial x$ and the vertical gradient image $\partial A/\partial y$ are given by:

$$\frac{\partial A}{\partial x}(x,y) = \sum_{-R \le i,j \le R} \frac{-i}{2\pi\sigma^4} e^{-\frac{i^2+j^2}{2\sigma^2}} A(x+i, y+j), \tag{3}$$

$$\frac{\partial A}{\partial y}(x,y) = \sum_{-R \le i,j \le R} \frac{-j}{2\pi\sigma^4} e^{-\frac{i^2+j^2}{2\sigma^2}} A(x+i, y+j), \tag{4}$$

where σ and R represent the standard deviation and the kernel size of the filter, respectively. We currently use $\sigma = 3$ and $R = 9$.

In summary, the single-image algorithm optimizes the cost function $G(I_F, I_D)$ with respect to the transformation $\mathbf{T}$. On the other hand, our biplane-image version optimizes the sum of two cost functions, each computed for one of the biplane images. The three procedures, namely 1) DRR generation, 2) gradient image generation, and 3) NCC computation, are repeated until finding a local optimum.

3 GPGPU: GPU as a Computational Engine

The original purpose of the GPU is to project 3-D polygonal objects on the 2-D screen. To accelerate this rendering task, the GPU employs a parallel architecture [4] that consists of two different programmable processors: vertex processors (VPs) and fragment processors (FPs), as shown in Fig. 2. Since FPs in modern GPUs provide much higher performance than VPs, most GPGPU implementations use FPs as a computational engine in the GPU [15].

Such implementations employ the stream programming model [16], which exploits the data parallelism inherent in the application by organizing data into streams and expressing computation as kernels that operate on streams. Streams here are usually stored as texture data on the video memory, which can be fetched to FPs. A kernel is

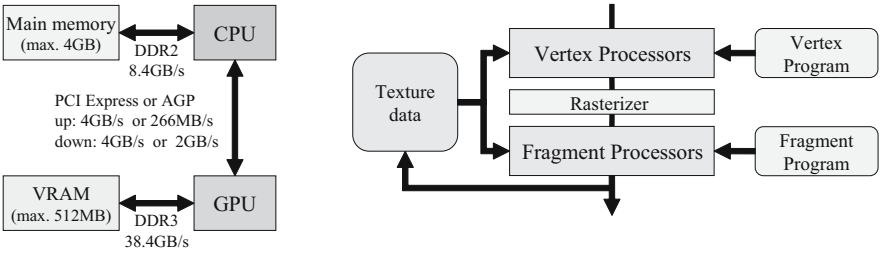


Fig. 2. GPU architecture

implemented as a FP program. The computed results can be transferred (readback) from the video memory to the main memory by using graphics APIs such as OpenGL [17].

In addition to the parallelization mentioned above, vectorization is also necessary to maximize the performance on the GPU. FPs support 4-length vector operations because they are designed to deal with pixels, which are four-component RGBA data representing red, green, blue colors and opacity. Since FPs apply vertor operations to a pixel, we must adapt data structure to obtain 400% speedup.

One concern about the GPGPU approach is that the GPU seems not be rigorous with computational errors [18], though it supports the IEEE floating-point representations [19]. Therefore, we should check computational results to verify if the error is acceptable.

4 2-D/3-D Registration on the GPU

Fig. 3 shows an overview of our GPU-based method. The key points of our design are as follows:

- (P1) Performance bottlenecks on the CPU should be implemented on the GPU with an algorithm suitable to the GPU architecture;
- (P2) The amount and frequency of communication between the CPU and the GPU should be minimized to achieve full acceleration on the GPU.

We think that the suitable algorithm mentioned above is an algorithm that (P1-a) resolves the target problem into a rendering problem, which is naturally accelerated by the GPU, or (P1-b) has fully data parallelism so that FPs can simultaneously process different pixels on the image (namely, texture), and if possible, with vector operations.

According to point (P1), we have decided to implement the three procedures on the GPU: DRR generation; gradient image generation; and NCC computation. These procedures take 99% of execution time on a sequential implementation.

1) DRR generation. As LaRose has presented in [10], this procedure can be naturally implemented on the GPU, because the principle of X-ray propagation is similar to that of object projection required for volume rendering. Therefore, we use a texture-based volume rendering method [20] for DRR generation in order to maximize the efficiency on the GPU. This method can be efficiently implemented on the GPU, because the texture-mapping and the alpha-blending procedures are hardware-accelerated on the GPU. Thus, our implementation for DRR generation satisfies point (P1-a).

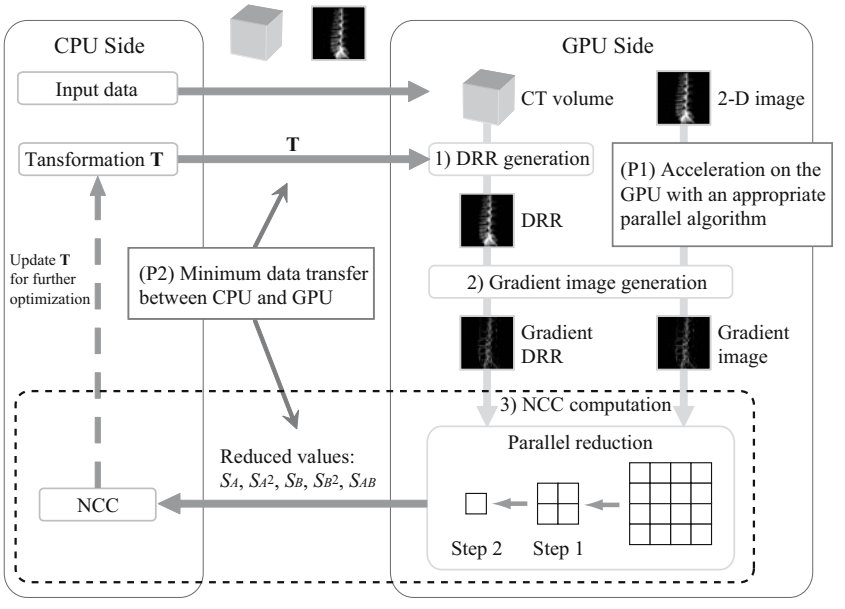


Fig. 3. Overview of 2-D/3-D registration on the GPU

Our method differs from LaRose's method in using 3-D textures instead of 2-D textures. As compared with 3-D textures, 2-D textures cannot produce higher quality DRRs, because 2-D textures cannot always be perpendicular to the view direction.

2) Gradient image generation. We implement a two-pass 1-D filter to reduce the time complexity of the 2-D filter:

$$P_{x1}(x,y) = \sum_j e^{-j^2/2\sigma^2} p(x, y+j) \tag{5}$$

$$P_{x2}(x,y) = \sum_i \frac{-i}{2\pi\sigma^4} e^{-i^2/2\sigma^2} p(x+i, y) \tag{6}$$

$$P_{y1}(x,y) = \sum_j \frac{-j}{2\pi\sigma^4} e^{-j^2/2\sigma^2} p(x, y+j) \tag{7}$$

$$P_{y2}(x,y) = \sum_i e^{-i^2/2\sigma^2} p(x+i, y) \tag{8}$$

This filter has fully data-parallelism, so that FPs in the GPU are allowed to simultaneously process different pixels in the image. Furthermore, vectorization can be applied to Eqs. (5) and (7) (Eqs. (6) and (8), also), because these computations (1) have no data dependence between them and (2) require pixels on the same location $p(x, y+j)$. Therefore, the horizontal gradient image and the vertical gradient image can be produced simultaneously by vectorization. To enable this, we use two of four (RGBA) components to process the 1-D filters at the same time. Our vectorization can be represented as follows:

$$\boldsymbol{P}_{xy1}(x,y) = \sum_j \left[\begin{pmatrix} e^{-j^2/2\sigma^2} \\ \dfrac{-j}{2\pi\sigma^4} e^{-j^2/2\sigma^2} \end{pmatrix} * \boldsymbol{p}(x, y+j) \right] \tag{9}$$

$$\boldsymbol{P}_{xy2}(x,y) = \sum_i \left[\begin{pmatrix} \dfrac{-i}{2\pi\sigma^4} e^{-i^2/2\sigma^2} \\ e^{-i^2/2\sigma^2} \end{pmatrix} * \boldsymbol{p}(x+i, y) \right] \tag{10}$$

where $\boldsymbol{P}_{xy1}$ and $\boldsymbol{P}_{xy2}$ represent two-component data containing pixels of gradient images after applying the first-pass filter and the second-pass filter, respectively, and $\boldsymbol{p}$ represent a vectorized pixel. Thus, our implementation for gradient image generation satisfies point (P1-b).

NCC computation. Finally, Eq. (2) indicates that there is no data-parallelism in NCC computation, because pixel values are merged into a single value (NCC). Therefore, a naive method may process this procedure on the CPU. However, this is not recommended from the viewpoint of (P2). That is, if the CPU takes the responsibility for NCC computation, we have to transfer the DRR from the GPU to the CPU at every optimization step. This communication may result in a lower performance, because the DRR is 2-D data. To tackle this problem, we decompose the computation into two parts: reduction operations on the GPU and the remaining operations on the CPU. This allows us to transfer only five floating point numbers, S_A, S_{A^2}, S_B, S_{B^2}, and S_{AB}, instead of the 2-D DRR. Then, the CPU computes NCC using these numbers according to Eq. (2). Thus, although parallelization cannot be applied to the entire computation, we can parallelize reduction operations with reducing the amount of communication between the CPU and the GPU.

Given an image of $n \times n$ pixels, the parallel reduction can be done in at most $\log n$ steps, as shown in Fig. 3. Although this sequence of steps must be serially processed,

```
void runReduction(Region &region, // Region for drawing
      TextureObject *rtexture, // Initialized texture object
      RenderTexturePBuffer *pbuffer, // Initialized pixel buffer
      BufferSpecifier &rtextureBuffer, // Input buffer, namely gradient images
      BufferSpecifier &drawBuffer) // Output buffer
{
      cgGLEnableProfile(vertexProfile and fragmentProfile);
      cgGLBindProgram(vertexProgram and fragmentProgram); // See Fig. 5
      glDrawBuffer(drawBuffer); // Specify output buffer
      rtexture->bindTexture(); // Bind texture
      pbuffer->bindTexImage(rtextureBuffer); // Bind input buffer
      glClear(GL_COLOR_BUFFER_BIT); // Clear output buffer
      glRecti(region); // Draw specified region
      glFlush(); // Flush issued OpenGL commands
      pbuffer->releaseTexImage(rtextureBuffer);
      cgGLDisableProfile(vertexProfile and fragmentProfile);
}
```

Fig. 4. CPU program for parallel reduction

```
// Data structure for passing coordinates data from VPs to FPs
struct ReductionCoords {
    float4 position : POSITION; float2 coord0 : TEXCOORD0; float2 coord1 : TEXCOORD1;
    float2 coord2 : TEXCOORD2; float2 coord3 : TEXCOORD3; float2 coord4 : TEXCOORD4;
    float2 coord5 : TEXCOORD5; float2 coord6 : TEXCOORD6; float2 coord7 : TEXCOORD7;
};
// Coordinates computation for parallel reduction of 3×3 pixels
ReductionCoords reductionVertex9(float4 position : POSITION, // Vertex coordinates in range [0.33, 0.66]
    uniform float4x4 modelViewProjMatrix : state.matrix.mvp) // Transformation matrix
{
    ReductionCoords output;
    output.position = mul(modelViewProjMatrix, position); // Update volume position
    output.coord0 = position.xy * 3 - 1.0f; // Adjust output range [0.33, 0.66] to input range [0, 1]
    output.coord1 = output.coord0 + float2(1.0, 0.0); output.coord2 = output.coord0 + float2(0.0, 1.0);
    output.coord3 = output.coord0 + float2(1.0, 1.0); output.coord4 = output.coord0 + float2(0.0, 2.0);
    output.coord5 = output.coord0 + float2(2.0, 0.0); output.coord6 = output.coord0 + float2(1.0, 2.0);
    output.coord7 = output.coord0 + float2(2.0, 1.0);
    // Address for pixel (2, 2) cannot be precomputed due to limited number of VP registers
    return output;
}
```

(a)

```
// Parallel reduction of 3×3 pixels
float3 reductionSum9RGB(ReductionCoords input,
    uniform samplerRECT sampRect : TEXUNIT0) : COLOR
{
    float2 coord8 = input.coord0 + float2(2.0, 2.0); // Address for pixel (2, 2)
    float3 output = texRECT(sampRect, input.coord0).rgb; // Fetch pixel (0, 0)
    output += texRECT(sampRect, input.coord1).rgb; output += texRECT(sampRect, input.coord2).rgb;
    output += texRECT(sampRect, input.coord3).rgb; output += texRECT(sampRect, input.coord4).rgb;
    output += texRECT(sampRect, input.coord5).rgb; output += texRECT(sampRect, input.coord6).rgb;
    output += texRECT(sampRect, input.coord7).rgb;
    output += texRECT(sampRect, coord8).rgb; // Reduction for (2, 2)
    return output;
}
```

(b)

Fig. 5. GPU programs for parallel reduction. (a) Vertex program and (b) fragment program reduce 3×3 neighbor pixels $(0, 0) - (2, 2)$ into a single pixel $(0, 0)$. Except for $(2, 2)$, all of coordinates addresses are precomputed by VPs instead of by FPs in order to reduce computational amount. See [22] for details.

each step can be parallelized according to point (P1-b). In our current implementation, we have empirically determined that each of FPs merges nine pixels into a single pixel at a step. Furthermore, we apply vectorization to reduction operations. That is, four of the five sums are computed at the same time.

Note here that Chisu also have presented parallel reduction in [11]. However, this method may suffer in computational (round-off) error, because it uses mipmap textures to compute averages of pixels at each step. Since this error increases with the number of steps, the amount of communication cannot be reduced into optimal five numbers in most cases. Thus, their method has a tradeoff relation between the communication amount and the computational error. In contrast, our method computes sums of pixels, preventing computational error. Therefore, the DRR is fully reduced at the GPU side without any errors.

According to the designs mentioned above, we have implemented the method using the C++ language, the OpenGL library [17], and the Cg (C for graphics) toolkit [21]. Fig. 4 and Fig. 5 show an overview of the CPU and GPU programs implemented

for the parallel reduction procedure. Basically, the remaining procedures also can be implemented in the same way.

In this example program, 3×3 neighbor pixels are merged into a single pixel. Before performing rendering operations, the CPU program in Fig. 4 binds a vertex program and a fragment program, which express how a pixel should be computed through the rendering pipeline. For example, the vertex program in Fig. 5(a) computes the coordinates of neighbors in order to pass them to FPs. Then, the fragment program in Fig. 5(b) receives the coordinates from VPs and fetches the corresponding pixels to reduce them into a pixel. These rendering operations are activated by glRecti() in the CPU program and are terminated by glFlush(). After this, glReadPixels() is called to transfer computation results from the video memory to the main memory.

5 Experimental Results

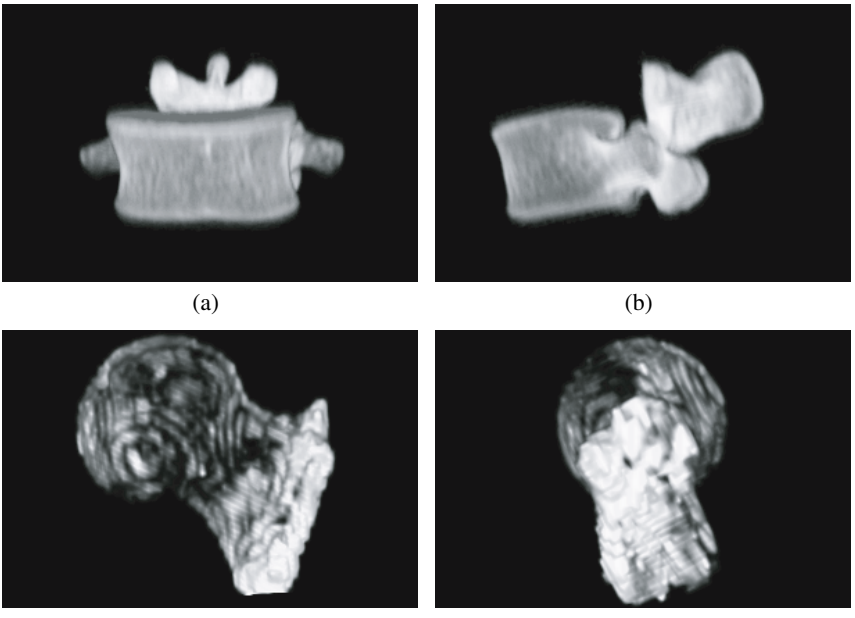
To evaluate the usability of our GPU-based method, we compare it with a cluster-based method [9] in terms of the registration time and the target registration error (TRE) [23]. The GPUs employed for experiments are summarized in Table 1. On the other hand, the cluster consists of 32 PCs each with a Pentium 3 1-GHz CPU. PCs are interconnected by a Myrinet 2000 switch [24], which yields a bandwidth of 2 Gb/s. Note here that the cluster-based method employ a ray casting method for DRR generation. Therefore, the base algorithm is slightly different from our GPU-based method.

Registration is performed using two datasets: the real spine and the femur phantom (see Fig. 6). Because our experiments focus on the comparison of implementation methods, we use DRRs instead of fluoroscopy images. That is, we first generate a DRR from a viewing point, and then use the DRR as an input image I_F to estimate the point from a randomly selected point.

Table 2 shows the timing results with breakdowns. Our GPU-based method completes a registration task within 10 seconds, which is permissible time for surgical assistances. We can also see that the method successfully reduces the time for DRR

Table 1. Experimental environment. Notations C_S and C_P are serial and parallel CPU environment, respectively. The remaining are GPU environments: G_1 is a laptop PC; G_2 and G_3 are desktop PCs with a previous generation GPU and a current generation GPU, respectively.

Environment	C_S	C_P	G_1	G_2	G_3
# of nodes	1	32	1		
CPU	Pentium 4 2.8 GHz	Pentium 3 1.0 GHz	Pentium M 2.0 GHz	Pentium 4 2.8 GHz	Pentium 4 3.4 GHz
GPU	—		Quadro FX Go 1400	Quadro FX 3400	GeForce 7800 GTX
Core clock (MHz)			125	350	430
Memory clock (MHz)			332	900	1200
Memory bandwidth (GB/s)			19.4	28.8	38.4
Fill rate (Gpixels/s)			2.2	4.2	10.3
Network	—	Myrinet 2000	—		



(a) (b) (c) (d)

Fig. 6. Biplane images of real spine and femur phantom datasets

Table 2. Timing results. Total time is the product of time per step and 300 steps.

Breakdown	Real spine V: 512 × 512 × 204 voxels ROI: 300 × 300 × 48 voxels I_F: 300 × 200 pixels					Femur phantom V: 256 × 256 × 367 voxels ROI: 54 × 38 × 55 voxels I_F: 300 × 200 pixels				
	C_S	C_P	G_1	G_2	G_3	C_S	C_P	G_1	G_2	G_3
DRR generation	2940	142	38.6	26.4	17.1	810	31	26.7	14.1	5.8
Gradient image	142	7	8.4	5.2	1.7	197	7	8.4	5.3	1.7
NCC computation	9	46	57.6	5.3	2.7	3	14	55.7	5.3	2.7
Reduction	—	—	7.9	4.9	2.4	—	—	7.9	4.9	2.4
Data transfer	—	—	49.6	0.2	0.2	—	—	47.7	0.2	0.2
Time per step (ms)	3091	195	104.6	36.9	21.5	1010	52	90.8	24.7	10.2
Total steps	300									
Total time	15 m	58 s	31 s	11 s	6 s	5 m	15 s	27 s	7 s	3 s

and gradient image generation, as compared with the sequential method (C_S). It also demonstrates further acceleration against the cluster-based method (C_P).

The parallel reduction for NCC computation reduces the amount of communication from 56 KB to 20 B. This reduction effect is significant for the laptop PC platform G_1, because such platforms do not have high-speed graphics bus between the CPU and the GPU. Even in desktop platforms, the data transfer time is reduced from 9 ms to 0.2 ms.

Table 3. Robustness results in terms of TRE (mm). Registration is done ten times for each. "Pass" represents the number of successful registration in ten trials.

Initial	Real spine						Femur phantom					
TRE	GPU single		GPU biplane		CPU biplane		GPU single		GPU biplane		CPU biplane	
(mm)	TRE	Pass	TRE	Pass	TRE	Pass	TRE	Pass	TRE	Pass	TRE	Pass
2– 4	2.37	4	0.10	10	0.27	10	7.05	0	0.53	10	1.13	8
4– 6	3.38	2	0.13	10	0.25	10	5.49	1	0.45	10	1.63	6
6– 8	3.98	3	0.09	10	0.24	10	6.04	1	0.51	10	3.27	2
8–10	6.84	0	0.09	10	1.70	9	7.18	0	0.94	9	12.71	1
10–12	5.10	3	1.46	8	3.12	8	8.48	0	0.68	9	9.34	0
12–14	7.48	0	1.11	7	8.92	4	6.63	0	0.73	9	11.19	1
14–16	12.41	1	5.87	4	7.25	5	6.91	0	1.86	9	15.18	1
16–18	13.73	2	7.09	4	13.73	2	7.09	0	5.37	5	16.46	0
18–20	19.88	0	11.13	2	13.17	3	7.19	0	4.83	5	22.63	0
20–22	10.84	1	10.71	2	18.48	1	8.72	0	6.58	5	22.23	0

Table 4. Robustness results obtained by the mipmap-based method [11]. In this experiment, we perform biplane registration with GPU-based DRR generation for the mipmap-based method, which originally performs single-image registration using the CPU-based DRR generation.

Intial	Real spine		Femur phantom	
TRE	TRE	Pass	TRE	Pass
2– 4	2.09	2	1.22	10
4– 6	1.85	1	1.17	10
6– 8	3.37	1	6.12	5
8–10	6.20	0	9.86	1
10–12	10.74	0	11.98	0
12–14	8.24	0	11.27	0
14–16	19.47	0	15.96	0
16–18	19.39	0	14.50	0
18–20	16.98	0	19.31	0
20–22	20.18	0	21.04	0

By comparing G_2 with G_3, we can see that the growth of GPU speed. That is, the current generation G_3 achieves almost double performance as compared with the previous generation G_2. This result is reasonable because GPU performance has doubled every six months [5].

We also investigate the robustness of our GPU-based method. Table 3 summarizes the alignment results. We repeat registration tasks ten times with different initial points. The registration is regarded as successful if the final TRE is less than 0.5 voxels, namely 0.66 mm for the spine data and 1.56 mm for the femur data.

As we can see in Table 3, our GPU-based method returns successful results if the initial TRE is less than 10 mm. There is no significant difference between the cluster-based method and the GPU-based method in terms of robustness. The results also show that using biplane images instead of a single image is necessary to obtain precise alignments in the depth direction.

Finally, we compare our method with a mipmap-based method [11], which we mentioned in Section 4. Table 4 summarizes the registration results. By comparing this table with Table 3, we can see that our method provides more robust results against the mipmap-based method. Furthermore, the registration performance of the mipmap-based method is almost the same as that of our method. Thus, we think that mipmap textures are not suited to parallel reduction due to computational error.

6 Conclusion

We have presented a fast 2-D/3-D registration method for biplane images using a GPGPU approach. Our method reduces registration time by eliminating performance bottlenecks on the CPU: DRR generation; gradient image generation; and NCC computation. Our method performs reduction operations on the GPU in order to minimize the amount of communication between the CPU and the GPU.

The experimental results show that our GPU-based method successfully completes a registration task in ten seconds. This timing result on the GPU is faster than that on the 32-node cluster. With respect to registration errors, the method demonstrates similar results as compared with CPU implementations. Thus, we think that our GPU-based method is useful for computer-aided surgery in terms of performance and robustness. As compared with the cluster-based method, we also think that the GPU-based method provides more attractive solution to medical doctors, because it needs less maintenance cost and less power consumption with higher fault tolerance.

Acknowledgments. This work was partly supported by JSPS Grant-in-Aid for Scientific Research on Priority Areas (17032007) and Scientific Research (B)(2)(18300009).

References

1. Lemieux, L., Jagoe, R., Fish, D.R., Kitchen, N.D., Thomas, D.G.T.: A patient-to-computed-tomography image registration method based on digitally reconstructed radiographs. Medical Physics **21**(11) (1994) 1749–1760
2. Hajnal, J.V., Hill, D.L., Hawkes, D.J., eds.: Medical Image Registration. CRC Press, Boca Raton, FL (2001)
3. Moore, G.E.: Cramming more components onto integrated circuits. Electronics **38**(8) (1965) 114–117
4. Montrym, J., Moreton, H.: The GeForce 6800. IEEE Micro **25**(2) (2005) 41–51
5. Fan, Z., Qiu, F., Kaufman, A., Yoakum-Stover, S.: GPU cluster for high performance computing. In: Proc. Int'l Conf. High Performance Computing, Networking and Storage (SC'04). (2004)
6. Galoppo, N., Govindaraju, N.K., Henson, M., Manocha, D.: LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. In: Proc. Int'l Conf. High Performance Computing, Networking, Storage and Analysis (SC'05). (2005)
7. Takizawa, H., Kobayashi, H.: Multi-grain parallel processing of data-clustering on programmable graphics hardware. In: Proc. 2nd Int'l Symp. Parallel and Distributed Processing and Applications (ISPA'04). (2004) 16–27
8. GPGPU: General-Purpose Computation Using Graphics Hardware (2005) `http://www.gpgpu.org/`.

9. Ino, F., Kawasaki, Y., Tashiro, T., Nakajima, Y., Sato, Y., Tamura, S., Hagihara, K.: A parallel implementation of 2-d/3-d image registration for computer-assisted surgery. In: Proc. 11th Int'l Conf. Parallel and Distributed Systems (ICPADS'05), Volume II Workshops. (2005) 316–320
10. LaRose, D.A.: Iterative X-Ray/CT Registration Using Accelerated Volume Rendering. PhD thesis, Carnegie Mellon University, Pittsburgh, PA (2001)
11. Chisu, R.: Techniques for Accelerating Intensity-based Rigid Image Registration. PhD thesis, Technische Universität München, München, Germany (2005)
12. Li, S., Pelizzari, C.A., Chen, G.T.Y.: Unfolding patient motion with biplane radiographs. Medical Physics **21**(9) (1994) 1427–1433
13. Penney, G.P., Weese, J., Little, J.A., Desmedt, P., Hill, D.L.G., Hawkes, D.J.: A comparison of similarity measures for use in 2-D–3-D medical image registration. IEEE Trans. Medical Imaging **17**(4) (1998) 586–595
14. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: NUMERICAL RECIPES in C: The Art of Scientific Computing. Cambridge University Press, Cambridge, UK (1988)
15. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Krüger, J., Lefohn, A.E., Purcell, T.J.: A survey of general-purpose computation on graphics hardware. In: EUROGRAPHICS 2005, State of the Art Report. (2005) 21–51
16. Khailany, B., Dally, W.J., Kapasi, U.J., Mattson, P., Namkoong, J., Owens, J.D., Towles, B., Chang, A., Rixner, S.: Imagine: Media processing with streams. IEEE Micro **21**(2) (2001) 35–46
17. Shreiner, D., Woo, M., Neider, J., Davis, T.: OpenGL Programming Guide. fourth edn. Addison-Wesley, Reading, MA (2003)
18. Hillesland, K.E., Lastra, A.: GPU floating point paranoia. In: Proc. 1st ACM Workshop General-Purpose Computing on Graphics Processors (GP^2'04). (2004) C–8
19. Stevenson, D.: A proposed standard for binary floating-point arithmetic. IEEE Computer **14**(3) (1981) 51–62
20. Cullip, T.J., Neumann, U.: Accelerating volume reconstruction with 3D texture hardware. Technical Report TR93-027, University of North Carolina at Chapel Hill (1993)
21. Mark, W.R., Glanville, R.S., Akeley, K., Kilgard, M.J.: Cg: A system for programming graphics hardware in a C-like language. ACM Trans. Graphics **22**(3) (2003) 896–897
22. Ikeda, T., Ino, F., Hagihara, K.: A code motion technique for accelerating general-purpose computation on the GPU. In: Proc. 20th IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS'06). (2006) 10 pages (CD-ROM).
23. Fitzpatrick, J.M., West, J.B., Maurer, C.R.: Predicting error in rigid-body point-based registration. IEEE Trans. Medical Imaging **17**(5) (1998) 694–702
24. Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.K.: Myrinet: A gigabit-per-second local area network. IEEE Micro **15**(1) (1995) 29–36

Author Index