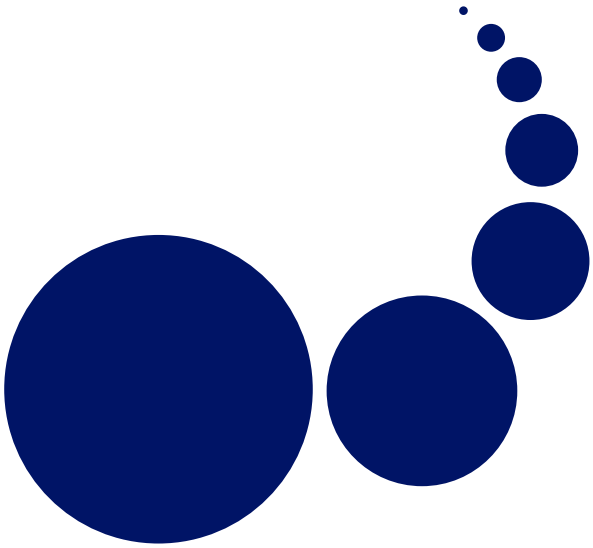
SCALABLE COMPUTING
Practice and Experience

Special Issue: Practical Aspects of Large-Scale Distributed Computing

Editor: Dana Petcu



Volume 8, Number 3, September 2007

ISSN 1895-1767

SUBSCRIPTION INFORMATION: please visit http://www.scpe.org

Scalable Computing: Practice and Experience

Volume 8, Number 3, September 2007

TABLE OF CONTENTS

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. i–v. http://www.scpe.org
ISSN 1895-1767

EDITORIAL: NEW FRONTIERS IN COMPUTATIONAL GENOMICS: IDENTIFICATION OF NOVEL FUNCTIONAL ELEMENTS AND GENOME-SCALE PATTERN ANALYSIS

Genomics researchers produce vast quantities of data that require detailed analysis. The amount of information makes it impossible to manually analyze the data. Thus, many bioinformatics software tools have been developed for the purpose of analyzing large-scale data distributed across numerous public data repositories. The discipline of bioinformatics, like the field of genomics, is in its infancy. This editorial illustrates this point by highlighting recent findings of the international ENCyclopedia Of DNA Elements initiative (referred to as the ENCODE project) [1], and by presenting exciting opportunities for computational genomics research.

1. Startling Outcomes of the ENCODE Project. The following quotation expresses the excitement that exists in the genomics community about the research area of identifying functional elements:

> The ENCODE project has set out to identify all the functional elements in the human genome. With the genome sequence now established, the next challenge is to discover how the cell actually uses it as an instruction manual.[1]

The metaphor of "an instruction manual" is useful for visualizing the challenges that exist. Below is a commentary on the recently discovered magnitude of the genomic instruction manual.

> Researchers of the ENCODE consortium have analysed 1% of the human genome. Their findings bring us a step closer to understanding the role of the vast amount of obscure DNA that does not function as genes.
>
> We usually think of the functional sequences in the genome solely in terms of genes, the sequences transcribed to messenger RNA to generate proteins. This perception is really the result of effective publicity by the genes, who take all of the credit even though their function is basically limited to communicating genomic information to the outside world. They have even managed to have the entire DNA sequence referred to as the 'genome', as if the collective importance of genes is all you need to know about the DNA in a cell.
>
> We should have guessed that this was merely prima-donna behaviour on the part of narcissist genes when the sequencing of the human genome revealed that they comprise only a small percentage of the DNA. And our confidence should have been shaken when some sequences located far from any genes were found to be strikingly conserved, indicating that they have some important function. Now, the ENCODE Project Consortium shows through the analysis of 1% of the human genome that the humble, unpretentious non-gene sequences have essential regulatory roles...
>
> The aim of the ENCODE (encyclopaedia of DNA elements) project is exactly that—to identify every sequence with functional properties in the human genome. The results of the pilot phase of this project, which involved an analysis of 1% (30 megabases) of the human genome, are not good news for genes, which will no longer be able to hog the limelight. Even this preliminary study reveals that the genome is much more than a mere vehicle for genes, and sheds light on the extensive molecular decision-making that takes place before a gene is expressed.[2]

First, it was discovered that much functional information is not "conserved" across organisms. (In the language of computer science, this means that functions and classes in different organisms have different source code implementations). With 5% of the human genome estimated to be under selective constraint [2], a large amount of genetic material has no attributed function. We know that 50% of the genome contains repetitive elements (considered nonfunctional), leaving 45% uncharacterized. To address the potential importance of these sequences new approaches are necessary to identify functional elements. Such knowledge will increase our abilities to understand and address problems that have genetic causes.

A second noteworthy result confirmed by the ENCODE project is that epigenetic marks predict the presence and activity of functional regions [3]. Histone proteins assemble into octameric complexes known as nucleosomes, around which the DNA wraps. The attractive force between positively charged amino tails of the histones and negatively charged phosphate backbone of the DNA creates a tightly associated structure. Modifications of the

[1] Decoding the blueprint (editor's summary), Nature 447 (14 June 2007).
[2] John M. Greally, Genomics: Encyclopaedia of humble DNA, Nature 447, 782–783 (14 June 2007)

histone tails, such as acetylation, neutralize the positive charge, create a loosened position of the DNA, and enable access of the proteins that regulate transcription. These modifications are epigenetic in nature—meaning they are not encoded in the DNA sequence itself. Of the ∼100 modifications that are possible [4], a subset of 3 events distinguish sites of active promoter regions [3]. An overlapping yet distinctive set of events occurs in enhancer elements [5]. These types of large-scale analyses support the approach of *de novo* predictions of promoter and enhancer regions based on their epigenetic properties [6]. Notably, some regions carrying these modifications exhibit functional activity, yet lack sequence conservation between species [1]. *This observation strongly indicates that epigenetic modification may be a better indicator of functional elements than sequence conservation.* Given the diversity of epigenetic events that are possible, knowledge of additional functional regions is likely to emerge from studies of other combinations of epigenetic changes. Large-scale experimental analyses will require computational assessment to define the correlations between these modifications and their functional purposes.

A third surprising outcome of the ENCODE consortium is the amount of the genome that is expressed as RNA. Protein-coding genes, pseudo-genes, noncoding RNA, and functional RNA, such as rRNA, tRNA, snoRNA and miRNA, all contribute to this category. Novel observations show that transcripts can routinely span multiple gene loci creating fusions of transcripts from neighboring genes [7]. These unique products have properties of both of the individual genes. Noncoding RNA transcripts have recently been found traversing intergenic regions, which were previously thought to be nontranscribed. Termed transcriptionally active regions (i.e., TARs or transfrags), these unexplainable transcripts have been identified in large numbers [8]. *Remarkably, up to 93% of bases in the ENCODE regions are transcribed—the noncoding transcripts are perhaps the best evidence that the genome still holds many elusive secrets.*

The definition of genes is evolving from the objects once identified as annotated open reading frames of a genomic sequence into a broader description to include expressed sequences. *The new definition can be summarized as "a union of genomic sequences encoding a coherent set of potentially overlapping functional products"* [7]. To be compatible with the former definition of genes, this definition encompasses the conventional aspect of individual transcripts from unique locations that encode protein or functional RNA. It now extends to include the union of sets of overlapping regions that produce similar yet distinct products. The definition furthermore removes alternative untranslated regions (5' and 3' UTRs), which are not encoded in the final protein product of a gene but may be alternatively utilized by various transcripts from the same gene. The requirement of a functional product also precludes most transfrags from this definition, simply because we have not identified their role in a majority of cases.

A fourth outcome is that regulatory binding sites (motifs) extend upstream and downstream of transcription start sites; this implies a need to redefine what constitutes a promoter region. Binding studies indicate that regulatory sequences surround transcription start sites (TSSs) and *are symmetrically distributed with no bias towards upstream regions.* Thus analyses of regulatory regions should also include sequences downstream of the TSSs. Furthermore, we can deduce that the downstream protein interaction sites are present in the 5'UTR sequences and first introns of many genes. As expected, these studies confirm that components of the basal transcription machinery such as RNA Polymerase II (POLII) and TATA Binding Protein Associated Factor 1 (TAF1) occupy active promoters and are depleted in inactive regions. It was previously thought that E2F1 was a specialized Transcription Factor (TF) (i.e., that it bound to Transcription Factor Binding Sites (TFBSs) found in only a small set of promoters). However, in studying these widespread patterns of promoter occupancy, the protein E2F1 was unexpectedly discovered at transcription start sites at a frequency similar to frequencies of generalized TFs such as TAF1 [9]. Thus, as more information becomes available, our rudimentary understanding of the process of transcription initiation will surrender to increasingly refined models of macromolecular protein complexes corresponding to specific promoter sequences that produce molecular responses to environmental conditions.

2. Research Frontiers. An important implication of the outcomes of the ENCODE project is that the existing models for computational genomics are inadequate. This section discusses specific research challenges that must be addressed to assimilate ENCODE's findings into computational genomics research and technology.

Exciting prospects. One percent of the genome was selected for the initial phase of the ENCODE Consortium analysis. The successful outcome of the project, providing a catalogue of functional elements and insight into a broader understanding of the content and organization of the human genome, paves the way for a larger, more comprehensive analysis stage. In scaling the ENCODE project from 1% of the genome

to 100%, all aspects of the project must expand. For a timely conclusion, experimental analyses need high-throughput approaches. Computational data storage, querying and retrieval must be scaled to larger sizes than ever before. Data analysis will require speed and capacity to handle immense amounts of data. The plethora of biological elements identified by the next phase of ENCODE will provide opportunities to study biological phenomena that have never been discovered. Data availability is only one component part. Important metadata, including conditions used for data collection are necessary for the correct interpretation of the data by external parties. Furthermore, recognition of the intrinsic noise produced by high throughput analyses must be emphasized to minimize false-positive predictions and the postulation of misguided biological hypotheses.

Integrative data analysis approaches. Integrative data analysis is emerging as a major theme for computational genomics. Characteristic features are present in classes of functional data, thereby creating a broader picture of how the individual parts create a functional entity. Recognition of these features is useful for supervised learning approaches. Features are often measured one experiment at a time. All sets of features can then be compared through an intersection of their overlapping positions in the human genome. Converting raw sequences into their genomic coordinates requires the use of alignment tools, such as BLAST or BLAT. Databases, which were established to help with comparison of high-throughput datasets, include the UCSC Human genome Table Browser [10], Galaxy [11], and ENCODEdb [12]. This interconnected network of databases allows each one to specialize in a specific area, while proving portability of data for comparison. One of the newer databases in this field, ENCODEdb serves as a prototype for the storage and retrieval of micro-array based data, which has been historically difficult to handle in search and retrieval operations. The ENCODEdb portal converts microarray data (either expression data or ChIP-chip data) into genomic coordinates for evaluation against other functional elements discovered in the genome.

Regulatory genomics. The question of what is regulatory in the genome remains unanswered. Former estimates of 5% of human sequence being under selective constraint may be an underestimate, if the neutral rate of substitution has been miscalculated [13]. The elements most commonly used as a model for the neutral rate of evolution are repetitive elements. These elements have long been considered "junk" or parasitic DNA. Their apparent lack of functional activity in the genome creates an assumption that these sequences diverge along a neutral model. However, some elements have been adapted for functional purposes and are under purifying selection [14, 15]. Sequence conservation present in this category of elements could contribute to an inaccurate assessment of the neutral rate. The prevalence of noncoding transcripts in the genome provides an avenue for speculating that a larger fraction of the genome is functional than previously thought. An upper estimate of 20% can be supported by quantitative studies sprinkled with some speculation [13]. Some researchers conjecture that the majority of the genome is functional [16]. Computational approaches [17] are helping to explore this issue.

On a smaller scale, studies of promoter regions in the human genome continue to reveal new information. For example, the co-occurrence of transcription factor binding sites corresponds to signals for tissue-specific expression, yet only a few patterns are known [18]. One controversial hypothesis that small RNA molecules activate promoter regions seems plausible yet heavily scrutinized [19]. Another area that is re-emerging is the study of bidirectional promoters [20]. Positioned between two genes, these regulatory elements control two genes rather than one. This association implies that mutations in a bidirectional promoter could negatively affect two genes rather than one. Quite often these promoters regulate transcription of DNA repair genes, several of which are known tumor suppressor genes [21]. This finding suggests that bidirectional promoters should be further examined for a role in cancer when their function is compromised.

Identification of ALL functional elements, interactions, and 2D/3D structural aspects of the human genome. The ENCODE project identifies 5 types of regulatory elements: promoters, enhancers, silencers, insulators and locus control regions. Additional categories that have yet to be fully elucidated include replication origins, structural features, and noncoding and anti-sense transcripts. One problem that looms well beyond the identification of regulatory elements is the determination of which genes they regulate. Regulatory elements can act over long genomic distances through mechanisms that are not fully elucidated, which may include looping out of intervening regions. Such long-distance interactions are difficult to predict with current techniques. The seemingly random placement of regulatory elements complicates the interpretation of which gene receives the activity; however identification of domains of active and repressed DNA may help to narrow the search [22]. Distant regulatory elements have been predicted through the use of sequence conservation or tandem arrangements of protein binding sites. The collection of high-throughput data representing epigenetic modifications

may thrust the study of regulatory regions past the annotation phase and into interpretation of biologically relevant effects. This tremendous amount of data will require foresight and planning.

Obviously, there is a need for new computer models and algorithms that incorporate the findings of ENCODE for the discovery of new genomic elements.

Another interesting computational problem relates to determining specific functions of genomic elements. Function is specific for each cell type; *therefore the possibility exists to define signatures for active sequences representing every cell type. Such analyses would capitalize on computational machine learning approaches.*

Numerous additional challenges provide prospects for future research. For example, the need to identify genome-wide patterns that reveal clues about the language of the genome will require new software. Ultimately, the syntax and semantics of genomes need to be discovered. Thus, there is a need for new computational approaches, which reveal unique aspects of the genome, such as the following:

1. Which portions of genomes are likely to have biological meaning?
2. What are the meanings?
3. Is there a vocabulary of the DNA? If so, what are the biological words, phrases, grammar, semantics, etc?

The answers to these questions will have significant impact on the future of biological research. They will contribute to a more complete understanding of the purpose of genomes and their composite elements. Subsequent analyses will provide datasets for wet-bench scientists to use in studying the functions of newly discovered genomic elements. Benefits also include gaining enough knowledge to predict which nucleotide-level perturbations contribute to genetic disease.

Given that we know so little about the composition of genomic sequences and that much more of the genome appears to be functional than was previously recognized, it is likely that new kinds of functional elements and associated mechanisms remain to be discovered.

Another aspect of functional elements is the role of secondary structure. For example, RNA molecules often form folded arrangements based on the composition of their complementary single stranded nucleotide interactions. Secondary structure is utilized in DNA during recombination events in DNA repair processes. Even promoters regions, such as the *MYC promoter,* form higher-order structures involved in regulatory function [23].

As structural features, physical associations, and protein affinities are interwoven into the story of gene regulation, we must also consider that linear order may be indispensable for proper genomic performance. Components that insulate elements from spurious interactions may not be detectable outside of their genomic context. Experiments creating rearrangements of sequences within their endogenous locations, termed chromosomal engineering, are simple only in conceptual terms; yet they offer the promise of accentuating the subtleties behind complex genomic regulation. Sequences that function to create the appropriate distance between genomic regions may perform a passive, albeit essential, function in the genome.

Yet another challenge is to extend such analyses to genomes other than human. The continual production of new genomic sequences provides opportunities to explore composition of sequences built on a similar model, yet designed to fulfill a different biological niche. The likeness of the sequences can be further studied through comparative genomics, which aims to discover similarities and differences among the patterns, syntax, semantics, and functional elements of various genomes. Approaches for discovering and visualizing functional genomic elements are emerging [17, 24, 25, 26], but many challenges remain.

Personalized medicine. Identifying regulatory regions in the human genome underscores a central goal in biology. That goal is the prediction of sequence changes that correlate with disease. Two complementary approaches may be necessary to accomplish this task. The first technique aims at finding all sequence changes that are likely to be involved in disease. The second technique is to find all regulatory regions whose disruption may lead to disease. Patterns associated with each approach could boost their effectiveness by narrowing the amount of data that must be evaluated. The PhenCode database provides one example of linking genotype (DNA sequence) and phenotype (characteristics) by mapping alterations in amino acids back to the DNA sequence of the human genome. This approach aims to record the patterns of change that result in disease [21]. As genomic sequencing becomes more cost-effective, applying the technology to each person's genome could become a reality. Such an era of personalized medicine will embrace computational analysis as an integral part of the process.

In summary, the current generation of bioinformaticians and molecular biologists will map the elements of the genomes, and will make discoveries that will help to cure many of the ills of mankind. Our success will be enabled by continuing to refine our paradigms to encompass outcomes of initiatives such as the ENCODE

project, and by developing a new generation of computational genomics software tools that can help biologists to deal with the complexity of the new paradigms.

Acknowledgments. LE is supported by the Intramural Research Program of the National Human Genome Research Institute, National Institutes of Health.

LW's research is supported by several programs of the Ohio University: 1804 Research Fund, Stocker Research Fund, Graduate Education and Research Board, Research Priorities Fund, and Stuckey Professorship.

REFERENCES

[1] Birney E., Stamatoyannopoulos J. A., Dutta A., Guigo R., Gingeras T. R., Margulies E. H., Weng Z., Snyder M., Dermitzakis E. T., Thurman, R. E., et al. (2007) *Nature 447,* 799–816.
[2] Chiaromonte F., Weber R. J., Roskin K. M., Diekhans M., Kent W. J. & Haussler D. (2003)
[3] Koch C. M., Andrews R. M., Flicek P., Dillon S. C., Karaoz U., Clelland G. K., Wilcox S., Beare D. M., Fowler J. C., Couttet P., et al. (2007) *Genome Res* **17,** 691–707.
[4] Bernstein B. E., Meissner A. & Lander E. S. (2007) *Cell* **128,** 669–81.
[5] Heintzman N. D., Stuart R. K., Hon G., Fu Y., Ching C. W., Hawkins R. D., Barrera L. O., Van Calcar S., Qu C., Ching K. A., et al. (2007) *Nat Genet* **39,** 311–318.
[6] Roh T. Y., Wei G., Farrell C. M. & Zhao K. (2007) *Genome Res* **17,** 74–81.
[7] Gerstein M. B., Bruce C., Rozowsky J. S., Zheng D., Du J., Korbel J. O., Emanuelsson O., Zhang Z. D., Weissman S. & Snyder M. (2007) *Genome Res* **17,** 669–81.
[8] Kapranov P., Willingham A. T. & Gingeras T. R. (2007) *Nat Rev Genet* **8,** 413–23.
[9] Bieda M., Xu X., Singer M. A., Green R. & Farnham P. J. (2006) *Genome Res* **16,** 595–605.
[10] Karolchik D., Hinrichs A. S., Furey T. S., Roskin K. M., Sugnet C. W., Haussler D. & Kent W. J. (2004) *Nucleic Acids Res* **32,** D493-6.
[11] Giardine B., Riemer C., Hardison R. C., Burhans R., Elnitski L., Shah P., Zhang Y., Blankenberg D., Albert I., Taylor J., et al. (2005) *Genome Res* **15,** 1451–5.
[12] Elnitski L. L., Shah P., Moreland R. T., Umayam L., Wolfsberg T. G. & Baxevanis A. D. (2007) *Genome Res* **17,** 954–9.
[13] Pheasant M. & Mattick J. S. (2007) *Genome Res* **17,** 1245–53.
[14] Krull M., Petrusma M., Makalowski W., Brosius J. & Schmitz J. (2007) *Genome Res* **17,** 1139–45.
[15] Bejerano G., Lowe C. B., Ahituv N., King B., Siepel A., Salama S. R., Rubin E. M., Kent W. J. & Haussler D. (2006) *Nature* **441,** 87–90.
[16] Slack F. (2006) *Genome Biology* **7,** 328. (A report from the meeting "Regulatory RNAs", the 71st Cold Spring Harbor Symposium on Quantitative Biology, Cold Spring Harbor, USA, 31 May-5 July 2006.)
[17] Rigoutsos I., Huynh T., Miranda K., Tsirigos A., McHardy A., & Platt D. (2006) *Proceedings of the National Academy of Sciences* **103(17),** 6605–6610.
[18] Pennacchio L. A., Loots G. G., Nobrega M. A. & Ovcharenko I. (2007) *Genome Res* **17,** 201–11.
[19] Check E. (2007) *Nature* **448,** 855–858.
[20] Lin J. M., Collins P. J., Trinklein N. D., Fu Y., Xi H., Myers R. M. & Weng Z. (2007) *Genome Res* **17,** 818–27.
[21] Yang M. Q., Koehly L. M. & Elnitski L. L. (2007) *PLoS Comput Biol* **3,** e72.
[22] Thurman R. E., Day N., Noble W. S. & Stamatoyannopoulos J. A. (2007) *Genome Res* **17,** 917–27.
[23] Belotserkovskii B. P., di Silva E., Tornaletti S., Wang G., Vasquez K. M. & Hanawalt P. C. (2007) *J. Biol Chem,* (in press).
[24] Meynert A. & Birney E. (2006) *Cell* **125(5),** 836–838.
[25] Xie X., Mikkelsen T. S., Gnirke A., Lindblad-Toh K., Kellis M., & Lander E. S. (2006) *Proceedings of the National Academy of Sciences* **104(17),** 7145–7150.
[26] Gu D., Lichtenberg J., Petri E., Welch J. D., Alam M., Nelson C., Ecker K., Nathaniel George, Josiah Seaman, Haiqang Zhang, Veronica Liang Wyatt S., & Welch L. R. (2007), `http://wordseeker.ath.cx` (The WordSeeker Functional Genomics Toolkit).
[27] Giardine B., Riemer C., Hefferon T., Thomas D., Hsu F., Zielenski J., Sang Y., Elnitski L., Cutting G., Trumbower H., et al. (2007) *Hum Mutat* **0,** 1–9.

Laura Elnitski
Genomic Functional Analysis Section
National Human Genome Research Institute, NIH
elnitski@mail.nih.gov

Lonnie R. Welch
School of Electrical Engineering and Computer Science
Biomedical Engineering Program
and Molecular and Cellular Biology Program, Ohio University
welch@ohio.edu

Scalable Computing: Practice and Experience
Volume 8, Number 3, p. vii. http://www.scpe.org
ISSN 1895-1767

INTRODUCTION TO THE SPECIAL ISSUE: PRACTICAL ASPECTS OF LARGE-SCALE DISTRIBUTED COMPUTING

In the exciting and rapidly moving world of parallel and distributed computing, advances in these multi-faceted fields need to be evaluated early and disseminated quickly. The International Symposium on Parallel and Distributed Computing (ISPDC), as well as Scalable Computing—Practice and Experience (SCPE), attempts to provide interactive, professional, and friendly forums for these purposes. Following the ISPDC meetings in Lille in 2005, Cork in 2004, Slovenia in 2003, Iasi in 2002, the 5th ISPDC meeting was held in Timisoara, Romania, from July 6th to 9th 2006. Of the 84 submissions received, 45 were chosen for delivery and inclusion in the IEEE Computer Press post-proceedings. These papers were limited to eight pages, thus the authors of the best papers were invited to publish extended versions of their contributions in SCPE. Nine cutting edge papers were selected for publication in this issue.

The first paper, by Emmanuel Jeanvoine, Louis Rilling, Christine Morin, and Daniel Leprince, presents Vigne, a Grid operating system that deals with the huge number of nodes and their dynamic behavior by using peer-to-peer overlays.

In the second paper, Ivan Frain, Abdelaziz M'Zoughi, and Jean-Paul Bahsoun introduce a dynamic quorum protocol, called the elementary permutation protocol, that permits the dynamic reconfiguration of a tree-structured coterie in function of the load of the machines that possess data replicas.

The third paper, by Ricolindo Carino and Ioana Banicescu, describes a dynamic load balancing tool intended to simplify the conversion of sequential programs containing computationally intensive one- or two-dimensional loops with independent iterates into parallel programs that execute with high efficiency on general-purpose clusters.

In the fourth paper, Miguel Bernabeu, Mariam Taroncher, Victor Garcia, and Ana Vidal describe a parallel implementation of a Lanczos-based method to solve generalized eigenvalue problems related to the modal computation of arbitrarily shaped waveguides.

Ouissem Ben Fredj and Eric Renault presents in the fifth paper of this issue the design of InfiniWrite, an implementation of the lightweight communication interface RWAPI over a Grid infrastructure called GRWA.

The sixth paper, by Keith Rochford, Brian Coghlan and John Walsh, describes the design and implementation of an agent-based architecture capable of detecting and aggregating status information using low-level sensors, functionality tests and existing information systems.

In the seventh paper, Xinfa Wei and Kaoru Sezaki propose a new multidimensional indexing structure for P2P systems called Distributed Hilbert R-trees that enables fault-tolerant, scalable, and multidimensional range query to be executed similarly as in overlapping regions tree in P2P systems.

Maria Lopez, Elisa Heymann and Miquel Senar analyze, in the eight paper of this issue, several heuristics used as static and dynamic scheduling strategies in a Grid environment and classify them according different practical criteria.

Finally, the ninth paper, by Rocco Aversa, Beniamino Di Martino, Salvatore Venticinque and Nicola Mazzoca, describes the functional model, the architecture design and the prototypal implementation of a platform called MAgDA, for services adaptation and delivery using agent technologies.

Note that the above mentioned papers represents different topics covered by ISPDC and SCPE that are intensively studied in the last years and we hope that the reader of this issue will find the papers interesting for his or her future research activities.

Dana Petcu

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 229–239. http://www.scpe.org
ISSN 1895-1767

USING OVERLAY NETWORKS TO BUILD OPERATING SYSTEM SERVICES FOR LARGE SCALE GRIDS

EMMANUEL JEANVOINE*, LOUIS RILLING†, CHRISTINE MORIN‡, AND DANIEL LEPRINCE§

Abstract. Using grid resources to execute scientific applications requiring a large amount of computing power is attractive but not easy from the user point of view. Vigne is a grid operating system designed to provide users with a simplified view of a grid. Vigne deals with the huge number of nodes in a large-scale grid and with the nodes' dynamic behavior by using peer-to-peer overlays as a keystone. In this paper, we show why it is highly desirable to use structured and unstructured peer-to-peer overlays for building the high-level services of Vigne grid operating system. To show the interest of our approach, we detail the features of two Vigne services built on top of peer-to-peer overlays. We also present experimental results obtained on the Grid'5000 testbed showing the scalability of the Vigne infrastructure based on overlays and its practical interest for the implementation of Vigne distributed services.

Key words. P2P overlay networks, grid operating system, application management, resource allocation

1. Introduction. Numerical simulation has an important place in several scientific fields where real experimentation is expensive or impossible. Many scientific applications require a large amount of resources that a single workstation cannot provide. Since the last decade, clusters and grids have become attractive for the execution of such applications. A challenge for the institutions needing a large computing power is to harness a huge amount of volatile resources distributed over many sites in a grid. As a first issue, the large number of resources and their physical distribution make it difficult to locate good resources. We propose to offer a *Single System Image* (SSI) view of the resources to overcome this issue. As a second issue, node additions, removals, and failures can occur frequently, which is hard for users and administrators to manage. Thus, we propose to design a self-organizing and a *self-healing* system to relieve users and administrators from large scale and dynamic behavior issues.

To simplify the use of a grid, we propose a grid operating system, called Vigne, designed with SSI, self-organizing and self-healing properties. To obtain these properties, we have based several services of Vigne on an infrastructure composed of decentralized self-healing overlays.

In this paper, we present the design principles of Vigne Grid operating system and demonstrate why it is highly desirable to use jointly structured and unstructured overlays in the Vigne infrastructure. The contribution of this paper is the following. We show how two kinds of overlays can be used to build two scalable, self-healing grid operating system services. The experimental results obtained executing Vigne on the Grid'5000 French national testbed show the scalability of the Vigne infrastructure and its practical interest. In particular, this infrastructure allows us to build a service to execute applications reliably on behalf of users, which originality is to be decentralized and self-healing.

The rest of this paper is as follows. In Section 2, we give an overview of the Vigne operating system and present its design principles. We explain the requirements of the Vigne services in terms of distributed object naming and location mechanisms in Section 3. In Section 4 we present the overlays used in the Vigne system and we detail two services based on these overlays. In Section 5, we present an experimental evaluation of the scalability of Vigne overlay infrastructure and provide preliminary performance results for the Vigne resource discovery service. We present related works in Section 6 and we finally conclude in Section 7.

2. Overview of Vigne. We are building the Vigne Grid Operating System (GOS) in order to relieve users and programmers from the burden of dealing with highly distributed resources over a set of volatile nodes. In this section we present the design principles we adopted to achieve this goal, and we give an overview of Vigne's services.

Design Principles. To design the services of Vigne, we followed three main principles. First, to provide users and programmers with *simple* abstractions of the distributed physical resources, the services of Vigne provide *Single System Image* (SSI). Users and programmers have the illusion to work on a single computer

*EDF R&D, IRISA PARIS project-team (emmanuel.jeanvoine@irisa.fr)
†Kerlabs (louis.rilling@kerlabs.com). This work was performed when Louis Rilling was affiliated to University of Rennes 1, IRISA PARIS project-team
‡INRIA Rennes-Bretagne Atlantique research centre, IRISA PARIS project-team (christine.morin@irisa.fr)
§EDF R&D (daniel.leprince@edf.fr)

that gathers the resources of all the nodes of the grid. This property is important because the distribution of the resources over a large number of nodes makes these resources very difficult to observe, select, and use for humans. This difficulty is all the more increased that these resources are shared between numerous users who run applications which characteristics vary greatly, and that the nodes hosting these resources are volatile.

Second, to ensure that the GOS is available and that applications run correctly despite the volatility of nodes, the services must be *self-healing*. Self-healing services tolerate several near-coincident reconfigurations and failures, and reconfigure themselves to tolerate reconfigurations and failures again. In particular, all services must be decentralized. This property is important to relieve administrators from complex system management when nodes are added or removed from the system or even fail, and to relieve users and programmers from dealing with these events when running and programming applications.

Third, to make the system scale to large numbers of nodes, the GOS services should content with *local knowledge* of the system state. This especially holds for the membership service, which is responsible for connecting the Vigne nodes. Indeed, maintaining a centralized global table of a large number of connected nodes despite the volatility of these nodes consumes too much bandwidth to run distributed applications efficiently.

The self-healing and local knowledge principles drive our choice of using peer-to-peer overlay networks as basic building blocks to design Vigne's services. In Section 3, we present how our three design principles induce requirements for basic object location mechanisms. These needs drive our choice of using jointly a structured and an unstructured overlay network.

Overview of Vigne's Services. As any operating system, a GOS virtualizes the physical resources to provide users and programmers with simple abstractions. To this end, Vigne is composed of high level distributed services comprising application scheduling, persistent data management, volatile data management [14], and of low level services comprising resources access control and membership. In this paper we present the application scheduling service of Vigne which abstracts computing resources. The two main mechanisms of the application scheduling service are application managers and a resource allocator. These mechanisms are built on a peer-to-peer-like infrastructure.

To abstract computing resources, the scheduling service runs each application under the control of a dedicated reliable agent called *application manager*. An application manager acts on behalf of the user to run efficiently the application and to ensure that it terminates correctly, despite nodes leaving the system or undergoing failures. To run the application efficiently, the application manager automatically selects the nodes on which the components of the application run, and moves components to nodes owning better resources when such nodes are discovered. Node leaves are handled by moving components running on these nodes onto other nodes. Node failures are handled by a configurable application fault-tolerance policy, which for example consists in restarting the application from a checkpoint. The benefit of overlay networks for application managers is presented in Section 4.3.

To select the nodes on which an application should run, the application manager uses a resource allocator based on scalable and *distributed resource discovery*. Given a description of needed resources, for example a number of CPUs of a given binary architecture and their minimum speed, the resource allocator discovers a set of suitable resources, and selects the most appropriate resources found. The resource discovery service of Vigne is presented in Section 4.2.

In Section 3 we draw requirements for these services in terms of basic object location mechanisms. We show that these requirements are met by self-healing and decentralized overlay networks.

3. Requirements in Basic Object Location. Since the services of a GOS handle objects that are distributed over the whole system, these services have to rely on distributed naming and location mechanisms for the objects they handle. In this section, we draw the requirements for these naming and location schemes from the design principles we have adopted to build Vigne. We show that these requirements are satisfied by overlay networks designed in recent peer-to-peer research.

3.1. Naming and Exact Location. A first class of requirements concerns location-independent naming of objects. We show that these requirements can be fulfilled using structured overlay networks.

Requirements. Building SSI services on top of a wide area distributed system needs location-independent global naming schemes. For instance, users should be able to get the status of their running applications without knowing which nodes run their applications. Similarly, users should be able to access files without needing to know which nodes host these files. Location-independent names also simplify the design of system services: location information about an object is isolated in a specific part of the service providing the object, and need

not be updated elsewhere in the system services. Location-independent names are all the more desirable that the nodes of the system are volatile.

Two reasons drive the need for location-independent names: objects may need to change their location, and objects may need to be replicated over several nodes. For persistent objects, like files, it is not reasonable to notify all nodes of the system of location changes because broadcast protocols do not scale well to high numbers of nodes.

Location changes are necessary for the high availability of objects. For instance, since nodes may leave the system at any time and should be able to do this without needing the approval of other nodes, long-running computations located on a leaving node should migrate to nodes remaining in the system in order to have a chance to complete.

Some objects need to be replicated on several nodes in order to remain available despite failures. For instance, users should be able to monitor and control their applications at any time. Therefore the system objects implementing these features should be highly available. Since network and node failures may occur at any time, replicating objects is the only way to keep them available. Depending on the replication scheme and on the object semantics, communications between an entity and a replicated object may involve all replicas of the object, or only a primary replica, or only the replica which is nearest to the entity. As a result, assigning a location figuring in the name of a replicated object may be a non-sense and may add complexity to fault-tolerance mechanisms when the location must change.

Finally, to enforce our design principles the naming schemes used by the GOS services should be self-healing and content with local knowledge of the system state.

Relevance of Structured Overlay Networks. The naming and exact location requirements can be fulfilled using structured overlay networks. Indeed, these overlays are self-healing and decentralized, and implement key-based routing (KBR). Using its structure, the overlay dynamically maps keys to nodes and routes a message to a key without that the sender needs to know the actual node to which the key is mapped. For this reason, the keys of a structured overlay network are good candidates to implement location-independent names.

To implement KBR, structured overlays connect the nodes of the system according to their names and the structure adopted. In Pastry [15], nodes have numerical names and are connected in clockwise order to build a logical ring. A key has a numerical name and is mapped to the node having the numerically closest name.

KBR is made efficient using shortcuts links, which typically allows an overlay to route a message in $\log(n)$ hops while keeping only $O(\log(n))$ links per node, where n is the number of nodes connected in the system.

Structured overlay networks can also help to build self-healing (highly available) objects using self-replication. Structured overlay networks are typically used to build fault-tolerant distributed hash tables (DHT). These DHTs store values on the nodes their keys are mapped to, and replicate the values on the neighborhood of these nodes in the overlay. For instance, DHTs based on Pastry self-replicate a value on the nodes preceding and following the node responsible for the key in the logical ring. This scheme can be generalized to make objects accessed using KBR self-replicating, and hence self-healing.

3.2. Attribute-Based Distributed Search. A second class of requirements concerns attribute-based search features. We show that these requirements can be fulfilled using unstructured overlay networks.

Requirements. Since we wish to provide SSI features for resource management, the GOS needs mechanisms to discover resources. Indeed, to run an application, the resource discovery service must find a suitable set of nodes in the grid. In contrast with the location and naming requirements shown in Section 3.1, the resource discovery mechanism must handle complex searches. Indeed, the system must find entities in the system without knowing their location or their name, but only using a description based on several criteria. Figure 3.1 shows an example of description used to search resources in a grid. It is important to note that a criterion is specified by a scalar value (like the CPU architecture or the operating system) or by a range of values (like the memory). As far as resource requirements for an application may be describe with such complex criteria, resource search cannot be performed with an exact location mechanism. With an exact location mechanism, the content of the queries must be hashed to produce a unique name. As far as attributes of a query may not have scalar values, it is very difficult to provide an hash function that conserve good load-balancing properties.

To deal with large scale, a centralized design must be avoided. Indeed, it could induce bottlenecks or single points of failure that would compromise the entire system. For the resource allocation service, a centralized database that contains all informations about existing resources in the grid would simplify the design of the system but we have to radically exclude it.

```
<resource_requirements>
  <cpu_architecture>X8_64</cpu_architecture>
  <nb_nodes>4</nb_nodes>
  <os>LINUX</os>
  <os_version>2.6.21</os_version>
  <cluster_scheduler>SH</cluster_scheduler>
  <network>MYRINET</network>
  <memory>2048-4096</memory>
</resource_requirements>
```

FIG. 3.1. *Example of description for resource search in a pseudo-XML synthax*

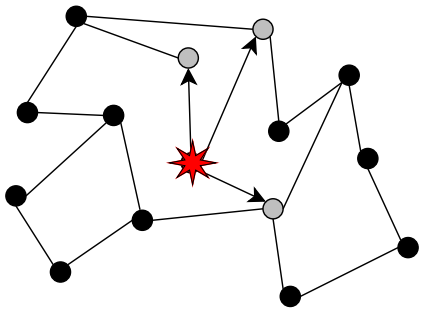


FIG. 3.2. *Flooding in an unstructured network, depth 1*

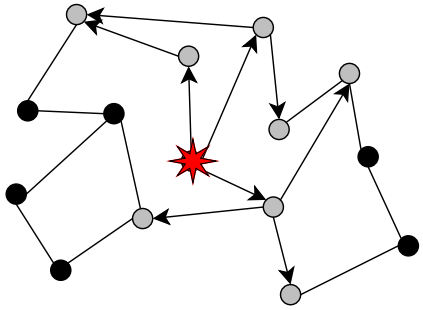


FIG. 3.3. *Flooding in an unstructured network, depth 2*

As a result, the GOS needs a distributed and fault tolerant search mechanism to locate resources in large scale environment just by knowing a set of constraints that must be fulfilled.

Relevance of Unstructured Overlay Networks. A distributed attribute-based search mechanism can be built using unstructured overlay networks like Gnutella [5]. As soon as a node joins the system, it establishes several connections with other nodes, called neighbors. In this type of overlay, the nodes do not take care of their neighbor names. However, nodes connections may take into account load balancing criteria or other policies.

To search an entity in the system, a node sends a query to its neighbors. If a neighbor hosts the searched entity, it replies to the requester. Otherwise it forwards the query to its own neighbors, and so on until a given depth. This depth is similar to the time to live (TTL) of packets in IP networks. This type of search is called flooding search. Figures 3.2 and 3.3 show respectively the first and the second step of a query propagated in an unstructured network with a flooding protocol.

As far as the cost of flooding the network might be prohibitive if the depth is high. Some optimizations like the random walks [17] have been studied by the peer-to-peer community. The random walks protocol works as follow. Instead of forwarding the query to all the neighbors like the flooding protocol, the query is forwarded to only one neighbor, randomly chosen. A random walk is also limited by a given TTL. So, when a random walk is finished (i. e. when the TTL is reached), if not enough results have been received, other random walks are initiated until a given timeout. If we consider an example where a node wants to find three resources of a king. Figure 3.4 shows the first step of a random walk protocol. At the issue of the first random walk, only one result has been received. So another random walk is performed, like it is shown on Figure 3.5. Two other results are obtained this time, so no more random walk is required.

TABLE 3.1
Pros and Cons of Structured and Unstructured P2P Networks

Type of overlay network	①	②	③
Structured	-	+	-
Unstructured	+	-	+

3.3. Discussion. Table 3.1 summarizes advantages and drawbacks of both structured an unstructured peer-to-peer networks. ① stands for the cost of neighborhood updates in term of bandwidth, ② stands for the

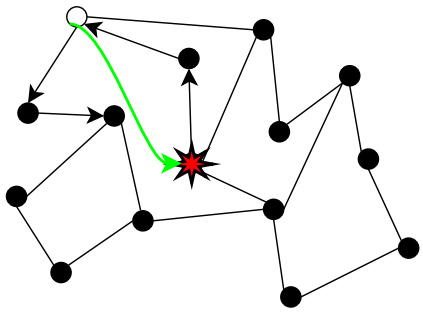


FIG. 3.4. *Random walks in an unstructured network, step 1*

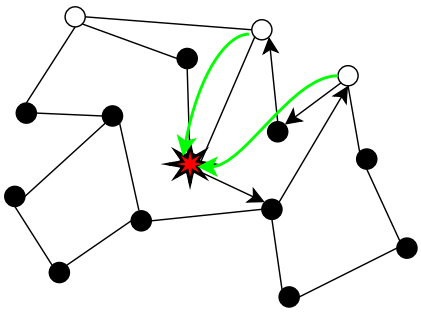


FIG. 3.5. *Random walks in an unstructured network, step 2*

capability to find a precise element in the network and ③ stands for the capability to make multi-criteria queries or range-queries.

Even if it exists substantial optimizations like [1], structured or unstructured network do not provide individually good properties for both exact location and complex search based on description. As a consequence, they are fully complementary.

4. Using Overlay Networks in Vigne. We present two services implemented in Vigne and based on the use of a decentralized peer-to-peer overlays: resource discovery and application management. We assume in the remainder that each node of the structured or unstructured overlay represents a node in the grid.

4.1. Vigne's Overlay Networks. In Section 3, we showed that Vigne's services require mechanisms for naming, exact location and attribute-based search. These requirements can be fulfilled using overlay networks. As a single kind of overlay cannot address all requirements, we choose to use two overlays as a basis for designing Vigne's services.

We showed that naming and exact location can be achieved using a structured overlay. For this reason, we implemented a structured overlay based on the Pastry routing algorithms [15], using the maintenance algorithms of Bamboo [13], with 256 bits naming spaces. Such naming spaces are large enough to identify nodes in the Grid or other objects like application managers and shared data objects. Furthermore, Pastry perfectly fits the requirements of reliable location of an entity in the system from any node and with a small number of hops.

We also showed that attribute-based search through a large scale system can be achieved using unstructured overlays. We have implemented an unstructured overlay whose membership protocol is based on Scamp [3].

On such an overlay, multi-attributes and range queries can be performed because all attributes are forwarded and can be analyzed on the flooded nodes.

Both structured and unstructured overlays have good properties regarding the local knowledge constraint. In Vigne's structured overlay, each node has 24 neighbors in the ring (12 preceding, and 12 following), and $O(\log(n))$ shortcut links (where n is the number of nodes in the system). The shortcut links allow the overlay to route messages to any node in $\log(n)$ hops on average. Furthermore, the number of neighbors known in the ring is large enough to prevent the system from partioning in most cases of simultaneous failures. In the unstructured overlay, each node has a neighborhood of $\log(n)$ nodes. Thus searches can be performed with a good cover rate and without using a large TTL (typically from 5 to 7).

Scalability is intrinsic to decentralized peer-to-peer overlays. Regarding unstructured overlays, basic flooding might be a performance issue but a Random Walk mechanism [17] would sidestep it. By design, both overlays have good self-healing properties.

In Vigne, we reduce the cost of using jointly structured and unstructured overlays by using a lightweight unstructured overlay that uses several properties of the structured overlay. Assuming that, the management of the unstructured overlay in case of churn is really simple and its cost is negligible. Thus, Vigne benefits of both type of peer-to-peer overlays at a low network bandwidth cost.

In Vigne, the structured and unstructured peer-to-peer overlays are complementary in their use and are keystones to build high level services.

4.2. Resource Discovery Service. One of the aims of Vigne is to execute applications without users needing to worry about the real execution location. To this end, the GOS must be able to find free resources in the grid to run applications.

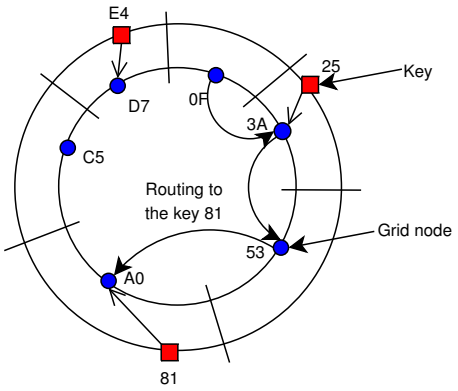


FIG. 4.1. *KBR example*

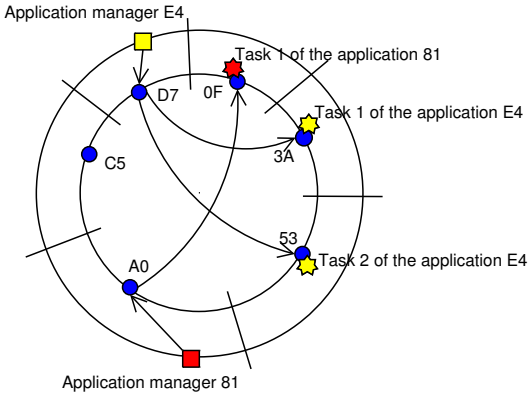


FIG. 4.2. *Grid example with two application managers*

Resource discovery consists in finding a set of resources corresponding to a specification provided by the user in order to run her application in good conditions. The specification includes several requirements of the application such as processor architecture, number of processors, local operating system, cluster scheduler, libraries, memory capacity, free disk space, etc.

These requirements like memory or free disk space are not scalar criteria. Furthermore an application may use a library whose version number is included in a specific range. Searches can be performed with a great expressiveness with an unstructured overlay. As a consequence, we decided to use an unstructured peer-to-peer overlay to perform resource discovery in Vigne.

We have implemented several protocols for resource discovery. The first one is a basic flooding protocol. The second one is a random walk protocol, less expensive than flooding in terms of bandwidth consumption. The third one improves the random walk protocol using a learning method to optimize the quality of results and bandwidth consumption.

4.3. Application Managers. In the Vigne operating system, the application management service is a crucial point in relation to applications life. The application management service manages the lifecycle of applications, and ensures that applications complete correctly despite node departures or failures. For instance, when the service detects the failure of a node where an application was launched, it can re-execute the application on another node. Since users should be able to interact with this service at any time and despite node departures and failures, this service must be highly available, and in particular self-healing.

The major contribution of our application management service is to decentralize application management and make it self-healing. To achieve this, we extensively use the structured overlay network. To build this service, overlay networks help to decentralize application management while still allowing users to access the service with the only knowledge of the identifiers of their applications, and help to make the service self-healing.

To distribute application management, the application management service introduces application managers, which are agents dedicated each to the management of one application. Application managers are distributed over the whole system using the key-based routing (KBR) scheme of the structured overlay network. Each application manager is assigned a key which is the SHA-1 hash of the (application binary MD5 sum, timestamp, name of host) triplet computed when launching the application. This triplet corresponds to the application identifier (AID). An application manager runs on the nodes its key is mapped to.

Figure 4.1 shows an example of key-based routing. In this case, the node 0F wants to send a message to the node responsible of the key 81. As a consequence, the message is routed to the node A0 since it is the node with the closer in the network identifier to 81.

Thanks to the KBR properties, an application manager can be reached by any entity of the system with the only knowledge of the corresponding AID. This property contributes to provide users and applications with single system image. For instance, if a user wants to know the progression status of the application execution, she routes a request to the SHA-1 hash of the AID.

A resource informs the application manager that the execution has completed using the same mechanism. Figure 4.2 shows a grid where two applications, respectively identified by the AIDs 81 and E4. One can see the two application managers and the tasks they monitor.

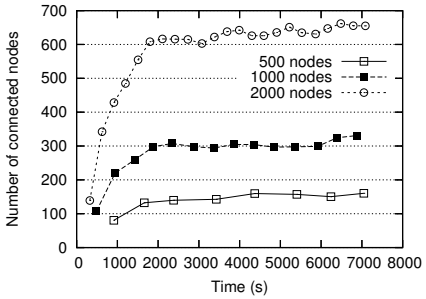


FIG. 4.3. *Number of nodes connected*

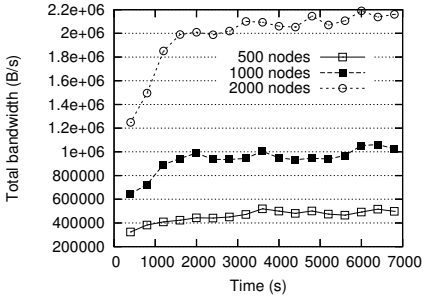


FIG. 4.4. *Global bandwitdh consumption*

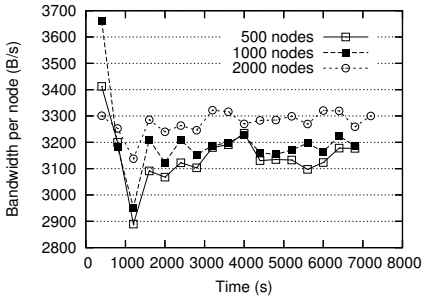


FIG. 4.5. *Bandwidth consumption per node*

Application managers are made self-healing using self-replication on the overlay network. More precisely, an application manager is made highly-available using active replication, and the locations of the replicas are automatically chosen in the overlay network using the same scheme as traditionally used in DHTs (see Section 3.1). Thus a constant degree of replication is automatically maintained in the logical neighborhood of the KBR target node for the application manager's key, which makes the application manager highly-available despite continuous node additions, departures, and failures.

5. Experimental Results. In order to validate the efficiency of Vigne, we conducted two kinds of experiments, both on the Grid'5000 French testbed [6]. Grid'5000 is the French national testbed for grid software where nodes are currently spread over eight sites.

5.1. Scalability of the Overlay Infrastructure. First we show that using decentralized self-healing overlay networks is reasonable and scalable. To do this, we measure the network bandwidth consumed for the maintenance of the overlays in systems composed of highly volatile nodes. This volatility represents an extreme case compared to the expected relative stability of the nodes composing an industrial grid. As a result, the bandwidth consumption observed in the experiment should represent a worst case.

We have used 100 computers belonging to Rennes' Grid'5000 site. Computers are bi-processor either AMD Opteron at 2.2 GHz or Intel Xeon at 2.4 GHz.

We emulated successively 500, 1000, and 2000 logical nodes. The logical nodes were mapped to the physical computers in a round robin fashion.

To measure the network bandwidth consumption, we measured the input traffic of each logical node.

Nodes arrivals and departures in the system have been simulated with two exponential laws. For each node, the mean time between two arrivals was 35 min (Poisson process), and the mean life time was 30 min.

Figure 4.4 and 4.3 respectively show the total bandwidth consumption and the number of nodes connected during the experiments. The bandwidth consumption observed confirms that emulating several nodes per computers does not hide congestion effects. If we take into account the total bandwidth divided by the number

of nodes, this ratio is constant whatever the number of nodes (Figure 4.5). Furthermore, the bandwidth consumption per node is reasonable with an average value of 3.3 KB/s.

Even if the dynamic behavior is hostile because peer-to-peer overlays have to re-configure them often, the system can scale regardless.

5.2. High Level Service Evaluation. We now evaluate the resource discovery service which is one of the high level services of Vigne. We show that this service, built on top of peer-to-peer overlays, is efficient.

We have used 472 computers belonging to four Grid'5000 sites (Bordeaux, Orsay, Rennes, and Sophia). Computers are bi-processor AMD Opteron at 2.0 GHz, AMD Opteron at 2.2 GHz and Intel Xeon at 2.4 GHz.

To produce an heavy load on a processor, we have used an iterative calculus code. Its execution time depends on the execution site.

TABLE 5.1
Computers dispersion over Grid'5000 sites and characteristics

Grid'5000 site	# of CPU	Execution time
Bordeaux	82	1740 s
Orsay	344	1959 s
Rennes (Paraci cluster)	98	1594 s
Rennes (Parasol cluster)	62	2689 s
Rennes (Paravent cluster)	198	2062 s
Sophia	160	1905 s
Total	944	1962 s

Table 5.1 shows the dispersion of processors across Grid'5000 sites. It shows also the execution time in seconds of the calculus code when the code is executed on a free processor, without concurrency.

The purpose was to evaluate the efficiency of the resource discovery service. To do that, we have measured the average execution time of the calculus process. If each of the 944 processes were simultaneously launched on 944 different processors, the average execution time of a process would have been 1962 seconds (see table 5.1). That corresponds to the execution time on a processor whose power would be the weighted mean of the powers of all the processors of our experiment.

TABLE 5.2
Average execution time according to the submission intervals

Submission interval	Average execution time	Overhead
1 s	2248 s	14.6%
2 s	2143 s	9.2%
5 s	2057 s	4.8%
10 s	2042 s	4%
20 s	2039 s	3.9%

To measure the efficiency of the resource discovery service of Vigne, we have performed five experiments where the task submissions have been spaced of 1, 2, 5, 10 or 20 seconds. We have not evaluated the global time of the execution of the 944 processes but the average time of a process execution. Thus, we have measured the ability for the resource discovery service to find the less loaded resources in order to execute the tasks.

Table 5.2 shows the average execution time for each experiment and the overhead of using our prototype compared to the average execution time of 1962 seconds. In these results, the larger the submission interval is, the smaller the execution time is. Indeed, if the submission interval is too small, the Vigne resource allocator is not able to evaluate accurately the load of the used resources. Thus, several processes may be executed on a single processor. In this case, the execution time is longer. However, the execution overhead is quite small regarding to the improvements in terms of automation or reliability. This overhead would be reduced with preemptive load balancing, because unbalances would be corrected after initial allocation. Furthermore, the time measured is larger than the real execution time. Indeed, detecting the end of an execution can last up to 60 seconds.

6. Related Work. We analyze related work in infrastructure choices, resource discovery and application management grid services.

6.1. Infrastructure. The architectural choices vary from centralized coordinators to fully decentralized and dynamic architectures.

BOINC-based projects and XtremWeb [2] target global computing using a centralized coordinator. This coordinator registers the resources ready to run some computations, and dispatches computations submitted by users. The coordinator is assumed to be stable and handles all the volatility of the resources. To achieve this stability, XtremWeb replicates the coordinator over a static set of nodes. These systems have proved to scale well in the context of the Internet but still impose a leader that hosts the coordinator, or a group of leaders hosting the coordinator's replicas. Moreover, the scalability of these approaches cannot be claimed for future use cases. In contrast, in our fully decentralized architecture one need not to assign any leadership to any node. Any node in the system can leave at will or even fail. Moreover, our local knowledge design principle is a stronger basis for scalability.

Most systems building scalable services use hierarchical approaches. For instance, Globus [4] extensively uses static hierarchies to build virtual organizations. Legion [11] uses a similar approach to name and locate objects. However, such architectures are very difficult to make self-healing, because the failure of one node in the hierarchy leads to the unavailability of all its sub-hierarchy. In particular, it is up to the administrators to define and repair these hierarchies. Moreover, without complex load balancing mechanisms, hierarchies suffer from contention in the high levels. In contrast, the overlay networks we adopted do not implicitly define hierarchies and tend to evenly distribute the load among the nodes.

GridOS [10] proposes to base naming, resource discovery, and resource allocation services on router-allocators, which are nodes that are organized in a similar way to routers in the Internet. The router-allocators of an organization maintain a global view of the organization's resources, and border allocators exchange summaries of their views between organizations. Like unstructured overlays, this mechanism is not well suited to access resources by location-independent names. Router-allocators could be made self-healing using the service mobility mechanism proposed by GridOS, but no automatic mechanism is provided to choose new nodes to run router-allocators.

Other works, including JXTA [18] and NaradaBrokering [12], aim at providing infrastructures to build high level peer-to-peer services. JXTA is built on an hybrid structured peer-to-peer network and provides a loosely consistent DHT, which model differs from our use of key-based routing. JXTA's DHT only stores advertisements for resources bound to peers. In particular, this DHT does not manage the location and the replication of the objects for which it stores advertisements.

NaradaBrokering provides a communication infrastructure including scalable event-delivery and publish-subscribe to build high level services. NaradaBrokering's features are complementary to the naming and resource discovery facilities we built. However, the brokering infrastructure's design assumes that a set of nodes remains relatively stable, and the volatility of the nodes is mostly considered for the clients of the brokering services.

6.2. Resource Discovery. XtremWeb [2] provides a resource discovery mechanism based on a centralized coordinator. To run a job, a client sends a query that describes requirements based on attributes to a coordinator. When a resource is idle, it asks to the coordinator for a job whose requirements are fulfilled by resource. A such resource discovery mechanism insures good efficiency but is not transposable to a decentralized system.

To provide a resource discovery mechanism, Globus uses the Monitoring and Discovery System (MDS) [4]. Resources informations are stored in the MDS by resources and are periodically refreshed. In a WSRF-compliant way, entities called information sources provide a Web service interface for communications with MDS. With up-to-date attribute-based queries to get location of suitable resources. Globus addresses the scalability issue by providing a hierarchical approach to MDS that uses *Agregators*. However, MDS neither provides self organizing nor a self-healing features. Indeed, resource administrators must say toward which MDS aggregator their resource is linked to.

Punch [9] uses Active Yellow Pages (ActYP) [16] to perform the resource discovery step resource allocation. ActYP is designed to optimize the response time of queries when queries are *similar*. ActYP is composed of one or more resource databases, a resource monitoring service that update regularly resource databases and a resource management pipeline. The resource management pipeline aggregates dynamically *similar* resources in order to optimize the response time of queries. Attribute-based queries are performed like classical queries in

a database. As far as databases are distributed, ActYP has good scalability properties. However, the databases of ActYP rely on stable resource, what is not considered in the design of Vigne.

In [7], authors study the convergence between peer-to-peer and grid communities. They propose a framework along four axes to design resource discovery architecture. Since they propose to perform resource discovery in a grid over an unstructured overlay network, this work is the closer to our for the resource discovery service. They evaluate random walk based protocols through a simulator they have implemented. We have chosen a such distributed scheme to design the resource discovery service in order to profit of the self-healing and self-organizing properties that are intrinsic to such overlay networks.

6.3. Application Management. Few projects include generic application management services to execute applications reliably. Chameleon [8] and XtremWeb [2] provide fault-tolerance mechanisms for a variety of programming models. In both systems, application management relies on a centralized entity (the main fault-tolerance manager in Chameleon, or the coordinator in XtremWeb), which is itself made reliable by replication on a static set of nodes. As a major contribution, our application managers decentralize application management, which is better to avoid contention and to resist to massive failures. Moreover application managers are replicated on dynamic sets of nodes, which allows them to adapt to any reconfiguration in the system.

7. Conclusion. This paper describes a general approach to build a self-healing scalable fully-decentralized SSI operating system for grids composed of a huge number of nodes. This paper brings two contributions. First we show that peer-to-peer overlays are a sound basis for building the distributed services of a GOS with the above properties. Moreover, we demonstrate that both structured and unstructured overlays are needed to meet the requirements of a GOS in terms of location-independent naming and of attribute-based distributed search. The cost of maintaining both kinds of overlay is reasonable as it is possible to take advantage of the structured overlay to maintain the unstructured overlay. We have built Vigne, a prototype of the proposed GOS. Experimentations carried out on the Grid'5000 grid platform demonstrate that the overlay infrastructure scales regardless the number of reconfigurations, the bandwidth consumption due to the overlay maintenance being very limited and constant whatever the number of nodes in the system. A resource discovery service and application managers have been implemented on top of the overlay infrastructure. Preliminary experiments show the efficiency of the resource discovery service. As a second contribution, our application managers make reliable application management decentralized and self-healing.

In future work we will experiment the resource discovery service with complex distributed applications, and we will experiment application managers in a fully dynamic system with various fault-tolerance policies to run applications reliably.

REFERENCES

[1] A. R. Bharambe, M. Agrawal, and S. Seshan, *Mercury: supporting scalable multi-attribute range queries*, in Proceedings of ACM SIGCOMM 2004, New York, NY, USA, 2004, ACM Press, pp. 353–366.
[2] S. Djilali, T. Herault, O. Lodygenski, T. Morlier, G. Fedak, and F. Cappello, *RPC-V: Toward fault-tolerant RPC for internet connected desktop grids with volatile nodes*, in Proceedings of the 2004 ACM/IEEE conference on Supercomputing, Pittsburg, PA, USA, Nov. 2004, ACM/IEEE, CS Press, p. 39.
[3] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, *Peer-to-peer membership management for gossip-based protocols*, IEEE Transactions on Computers, 52 (2003).
[4] *Globus.* Web Page: `http://globus.org/`
[5] *Gnutella.* Web Page: `http://www.gnutella.com/`
[6] *Grid'5000.* Web Page: `http://www.grid5000.fr/`
[7] A. Iamnitchi, I. Foster, and D. Nurmi, *A peer-to-peer approach to resource location in grid environments*, in Proceedings of HPDC 2002, Aug. 2002, p. 419.
[8] Z. T. Kalbarczyk, R. K. Iyer, S. Bagchi, and K. Whisnant, *Chameleon: A software infrastructure for adaptive fault tolerance*, IEEE Transactions on Parallel and Distributed Systems, 10 (1999), pp. 560–579.
[9] N. H. Kapadia and J. A. B. Fortes, *Punch: An architecture for web-enabled wide-area network-computing*, Cluster Computing, 2 (1999), pp. 153–164.
[10] K. Krauter and M. Maheswaran, *Architecture for a grid operating system*, in Proceedings of the First IEEE/ACM International Workshop on Grid Computing, vol. 1971 of Lecture Notes In Computer Science, Bangalore, India, Dec. 2000, Springer-Verlag, pp. 65–76.
[11] M. Lewis and A. Grimshaw, *The core Legion object model*, in Proc. of HPDC 1996, IEEE, CS Press, Aug. 1996, pp. 551–561.
[12] S. Pallickara and G. Fox, *NaradaBrokering: A middleware framework and architecture for enabling durable peer-to-peer grids*, in Proceedings of Middleware 2003, vol. 2672 of Lecture Notes in Computer Science, Springer, 2003, pp. 41–61.
[13] S. Rhea, D. Geels, T. Roscoe, and J. Kubiatowicz, *Handling churn in a DHT*, in Proceedings of the USENIX Annual Technical Conference, Boston, MA,USA, June 2004, USENIX, pp. 127–140.

[14] L. Rilling and C. Morin, *A practical transparent data sharing service for the grid*, in Proceedings Fifth International Workshop on Distributed Shared Memory (DSM 2005), vol. 2, Cardiff, UK, May 2005, pp. 897–904. Held in conjunction with CCGrid 2005.
[15] A. Rowstron and P. Druschel, *Pastry: scalable, decentraized object location and routing for large-scale peer-to-peer systems*, in Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware), Nov. 2001.
[16] D. Royo, N. H. Kapadia, J. A. B. Fortes, and L. D. de Cerio, *Active yellow pages: A pipelined resource management architecture for wide-area network computing*, in Proceedings of HPDC 2001, Washington, DC, USA, 2001, IEEE Computer Society, p. 147.
[17] F. Spitzer, *Principles of random walks*, Springer-Verlag, New York, 1976.
[18] B. Traversat, M. Abdelaziz, and E. Pouyoul, *Project JXTA: A Loosely-Consistent DHT Rendezvous Walker*, Mar 2003. `http://www.jxta.org/docs/jxta-dht.pdf`

Edited by: Dana Petcu
Received: May 29, 2007
Accepted: June 17, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 241–248. http://www.scpe.org
ISSN 1895-1767

HOW TO ACHIEVE HIGH THROUGHPUT WITH DYNAMIC TREE-STRUCTURED COTERIE *

IVAN FRAIN, ABDELAZIZ M'ZOUGHI, JEAN-PAUL BAHSOUN†

Abstract. Data replication permits a better network bandwidth utilization and minimizes the effect of latency in large-scale systems such as computing grids. However, the cost of maintaining the data consistent between replicas may become difficult if the read/write system has to ensure sequential consistency. In this paper, we limit the overhead due to the data consistency protocols by introducing a new dynamic quorum protocol called the *elementary permutation protocol*.This protocol permits the dynamic reconfiguration of a tree-structured coterie [2] in function of the load of the machines that possess the data replicas. It applies a tree transformation in order to obtain a new less loaded coterie.This permutation is based on the load information of a small group of machines possessing the copies. The implementation and the evaluation of our algorithm have been based on the existing atomic read/write service of [14]. We demonstrate that the elementary permutation ameliorates the system's throughput upto 50% in the best case. The results of our simulation show that the tree reconfiguration based on the elementary permutation is more efficient for a relatively small number of copies.

Key words. data replication, dynamic quorums, tree coterie, grid

1. Introduction. Replication permits a better bandwidth usage of the network by avoiding unnecessary data transfers between the sites. Nevertheless, high latency time exposes the replica management protocols to potential performance degradations. Among the existing replica management protocols, the quorum ones are well suitable because of their ability of diminishing the number of exchanged messages for the Read/Write operations applied to the copies. To perform an operation, copies of a quorum (read or write) must be contacted to insure consistency among the replicas. The set of all the possible quorums is called a coterie [11]. We can make a distinction between majority-based quorum systems [12, 19] and structured quorum systems [2, 13, 8, 15]. The former uses the majority of the replicas (possibly weighted) to construct the quorums. The latter uses a logical organization of the copies to diminish the quorum's cardinality and thus the number of exchanged messages of an operation.

Many works focus on quorum systems' performance improvement. They usually concentrated on the latency between processors that maintain the copies to construct adapted structured-coterie [20, 10, 6]. The authors of [7] proposed an algorithm for the creation of geographic quorums. They created a coterie in such a way that the distance between any client and any quorum is optimal. Their solutions are based on the distance between the sites, which is a static value and is related to the used accessing media's physical time latency and not to the loads of the machines or the network. There is a dynamic characteristic that must be taken into more consideration than the static one, which is the load of the processors [5]. We generally associate this load to the service response time of an operation. As the load increases, the service response time becomes longer.

In distributed environments like computing grids [9], the grid scheduler can not have total control over the nodes to which it delegates tasks. In fact, a shared machine in the grid is not always dedicated to the task that the scheduler has granted to it. The local user of the machine is its only master and hence he can ask it to realize tasks of which the grid system has no knowledge about them and that it can not quantify (in terms of the load) in advance. Moreover, computing grids are characterized by certain common properties such as weak bandwidth and high latency between the sites, distinct administrative domains and strong heterogeneity among the resources. Therefore such an environment is the perfect context to manage replicated data and use structured quorum consensus protocols based on the processor's load.

The problem to which this paper addresses is the dynamic reconfiguration of a tree-structured coterie in function to the load of its processors. A processor can be a node, a personal computer, or a single storage resource in a grid environment. What is important is the fact that a processor has a quantity of work to fulfill that we characterize it as its load and possesses a replica.

In this paper, we present a new tree-based coterie reconfiguration scheme used in a multi-reader/writer fault-tolerant algorithm. Our main contributions are:

1. The definition of quorum and coterie loads in order to construct a coterie based on the processors' loads.

*This work is part of the french RNTL project ViSaGe which is supported by the french ministry for research n°04K459.
†Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier 118, route de Narbonne, 31062 Toulouse Cedex 9, France {frain,mzoughi,bahsoun}@irit.fr

2. The introduction of a new reconfiguration scheme to the tree-structured coterie of [2]. This reconfiguration is based on the processors' loads and permits to diminish the coterie's overall load.
3. The extension of the algorithm of [14] to embed our elementary permutation. The extension is made to take into account the following three policies: an information policy, a selection policy and a reconfiguration policy. The information policy is used to collect the processor's load. The selection policy is used to choose the right moment of reconfiguration. The reconfiguration policy applies one or several above mentioned elementary permutation.
4. The implementation of this extended algorithm in the neko simulator [21] to demonstrate the performance improvement of our solutions. We show that throughput is improved of 50% by the elementary permutation under certain circumstances.

Related Work. There exists extensions of the different quorum protocols that permit the reconfiguration of a coterie when one of the nodes crashes (crash-stop) [1, 17, 3, 16]. When the failure of one node is detected, a new coterie is constructed with n-1 nodes. In our solution, in addition to being active or not, we take into account the availability of a node on its load basis. In [4], they focus on byzantine quorum systems and discuss on the quorum's load. However, their load definition of quorums differs from that of ours which will be given in the coming sections. In fact, the authors consider the inherent load of the coterie by taking into account the structure of the coterie and the accessing probability of a quorum but not the workload of each node.

In [18], the ViSaGe project was presented. This grid level software's objective is to provide to the grid community a flexible storage virtualization service. ViSaGe will permit to share storage resources in a transparent manner and with some levels of quality of services. An administrator of such a service can choose to plug any consistency management protocol such as the protocols we introduce in this paper.

The rest of the paper is organized as follows. In the following section, we present our load model. Sections 3 introduces the elementary permutation scheme. Section 4 presents the extension of the used read/write algorithm as well as the three policies that are introduced to integrate our permutation to this algorithm. Section 5 presents the implementation and the evaluation of performance of our proposal. The conclusion is the subject of section 6.

2. Model. We consider P_r as the set of all processors such that $P_r = \{P\, is\, a\, processor\}$. To each processor P, a working load is associated which will be denoted by x_p. Each processor possesses a copy of the data item d. In the remaining of this paper, we will reason about only a single data item, without losing generality.

Whatever is the quorum protocol type, either a majority quorum or a structured one, all of these types are subject to two properties : the intersection and minimality properties whose definitions are given hereafter [11].

Definition 2.1. ***Coterie and quorum***
Let C be a set of groups of Pr, then C is called a ***coterie*** *if it satisfies the following condition:*

$$C = \{Q \in \mathcal{P}(Pr) | \forall Q' : Q' \in \mathcal{P}(Pr) \wedge Q' \neq Q \rightarrow Q \cap Q' \neq \emptyset \wedge Q \nsubseteq Q'\}$$

The $Q \cap Q'$ property is called the intersection property and the $Q \nsubseteq Q'$ property is called the minimality property. Each element Q of a coterie C is called a ***quorum****.* The dynamic reconfiguration algorithm of a coterie that we present in the following sections is based on the load level of the processors to decide whether to perform a reconfiguration or not. One of the most important property that we take into account is the load of a quorum.

Definition 2.2. ***Load of a quorum***
The load Y_Q of a quorum Q is the maximum *of the loads x_P of the processors P that constitute this quorum.*

$$Y_Q = Max(x_P : P \in Q)$$

We consider that the accesses to different quorums of a coterie are fairly distributed among the quorums. We define the fairness access as such:

Definition 2.3. ***Fairness access***
Let m be the number of quorums Q of a coterie C. Let R_Q be the accessing probability to a quorum Q for a Read or Write operation. Then we consider the following:

$$\forall Q \in C : R_Q = \frac{1}{m}$$

We define the load of the coterie below. It will permit us to evaluate the efficiency of a coterie with respect to another, for the same number of quorums and for the same loaded nodes.

DEFINITION 2.4. ***Load of a coterie***
We denote the load of the coterie by δ_C which is equivalent to the sum of the loads of all the quorums of C.

$$\delta_C = \sum_{Q \in C} Y_Q$$

The quorum protocol that we use in this work is the one that was presented in [2]. In the remaining of this paper, when we use the word coterie, we mean a binary tree-structured coterie.

DEFINITION 2.5. ***Binary tree-structured coterie***
The processors are logically organized in the form of a binary tree. The processors are the nodes or the leaves of the tree. A Read or Write operation is carried out on a quorum of the coterie. A quorum is obtained by taking all the processors located on any path that starts from the root and terminates at the leaves of a binary-tree.

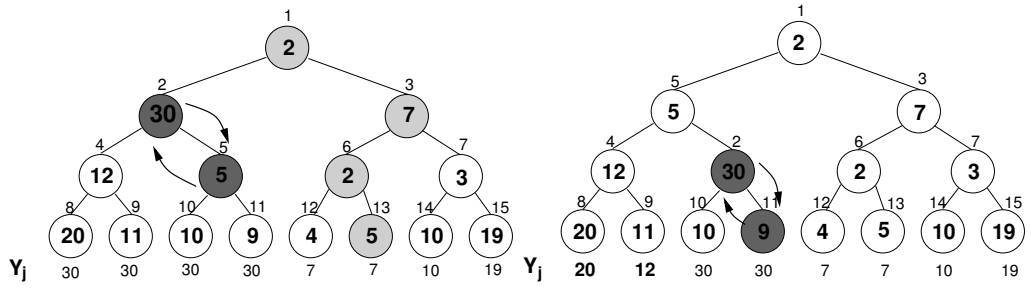
The binary tree protocol is classified as one of the structured quorum protocols. Intersection and minimality properties are well respected by this protocol. Figure 3.1 presents an example of a binary tree-structured coterie. In this figure, there are 15 processors that contain the replicas. The in-circle numbers represent the load of the processors and the out-circle numbers represent the identity of the processors. For example, P_1 is the root of the tree and its load is 2. The light gray-colored processors $\{P_1, P_3, P_6, P_{13}\}$, form one of the eight possible quorums. In the original paper [2], the tree quorum protocol was presented with a recursive definition of quorum which take care of faulty-processes. For example, if the root of the tree crashes, a quorum will be composed of two paths from the root to the any leaves in the right **and** the left sub-trees. This definition of quorum does not degrade gracefully when processors failed so we use the extension of this protocol presented in [16] which only use the paths from the root to the leaves to be a valid quorum. If a processor fails, a new coterie is constructed.

The Problem. is to minimize the tree-structured coterie's load. This can be achieved by applying a reconfiguration to a given coterie to obtain a less loaded coterie. Next, we propose the elementary permutation and we show it diminishes the coterie's load.

3. The Elementary Permutation. In this section, we define the notion of an *elementary permutation* that can be used to reconfigure a tree-structured coterie. In fact, during the dynamic reconfiguration of a coterie with partial knowledge of the load of the processors, a new coterie is constructed by applying one or several elementary permutations to the previous one (see section 4.2.1).

3.1. Principle and Algorithm. The principle of an elementary permutation algorithm is made up two steps:

1. finding a particular pattern in the tree of the form (P_a, P_b) such that P_b is the son of P_a and P_a's load is greater than P_b's load ($x_{P_a} > x_{P_b}$, see Figure 3.1).
2. if such a pattern was found, transforming it into another one (by permuting the two nodes thus P_a becomes the son of P_b) in such a way that it ameliorates the performance.



(a) $\delta_C = 163$ (b) $\delta_C = 135$

FIG. 3.1. Elementary permutation sequence

Figure 3.1 illustrates the application of several elementary permutations to a coterie. The algorithm 1 presents a more precise definition of an elementary permutation. In this algorithm, we also introduce a node and a binary tree coterie data types.

Algorithm 1: The Elementary Permutation Algorithm

```
type Node is record (name:String;load:Int)
type TreeCoterie is array(1..N) of Node
/* The first item of a TreeCoterie tc is tc[1] and corresponds to the root of the tree.
The left child of tc[i] corresponds to tc[2i] and the right child to tc[2i+1] */
Input: c:TreeCoterie,child:Int
Output: c':TreeCoterie
Data: nodeTemp:Node
begin
    c'←c
    if c'[child].load < c'[⌊ child/2 ⌋].load then
        /* Permutation between the parent and the child */
        nodeTemp←c'[child]
        c'[child]← c'[⌊ child/2 ⌋]
        c'[⌊ child/2 ⌋]← nodeTemp
    return c'
end
```

3.2. About Coterie's Load. By applying an elementary permutation to the tree, the performance must be ameliorated. The metric that we have taken to measure the gain in performance is the overall load of the coterie (Definition 2.4). An elementary permutation must at best diminish this load and at worst must not increase it.

Given two coterie configurations C and C' such that C' is obtained by applying an elementary permutation to C. We consider δ_C and $\delta_{C'}$ as the loads of the coteries C and C' respectively. If we consider the definition of the elementary permutation to be the same as defined previously (algorithm 1), then we must have $\delta_{C'} \leqslant \delta_C$.

Let us consider the levels of the nodes of the tree in the following manner: the nodes at the leaves are at level 0 and the root's node is at the highest possible level. According to the tree-structured coterie definition, we deduce that a node at level i belongs to 2^i quorums. An elementary permutation is applied to two nodes, the parent P_a^{i+1} at level $i+1$ and its child P_b^i at level i, if and only if $x_{P_a^{i+1}} > x_{P_b^i}$. Thus after the permutation, the more loaded node P_a^{i+1} will be at level i, hence we denote it by P_a^i whereas the less loaded node P_b^i, will be at level $i+1$, hence we denote it by P_b^{i+1}. So P_a^i will be contained in 2^i quorums whose loads remain the same and P_b^{i+1} will be in 2^{i+1} quorums distributed in the following manner:

- $(2^{i+1} - 2^i)$ quorums whose loads may have diminished because $x_{P_a^{i+1}} > x_{P_b^i}$ (the dark-gray colored left sub-tree of Figure 3.2)
- the other 2^i quorums of P_a^i (the light-gray colored sub-tree of Figure 3.2)

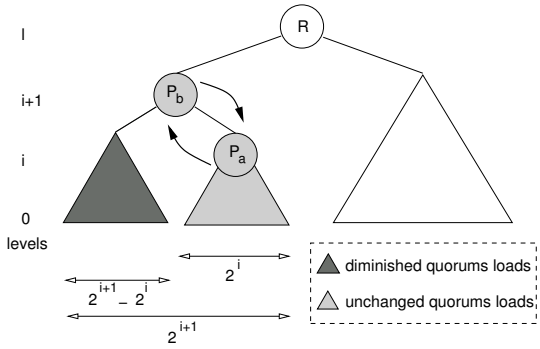
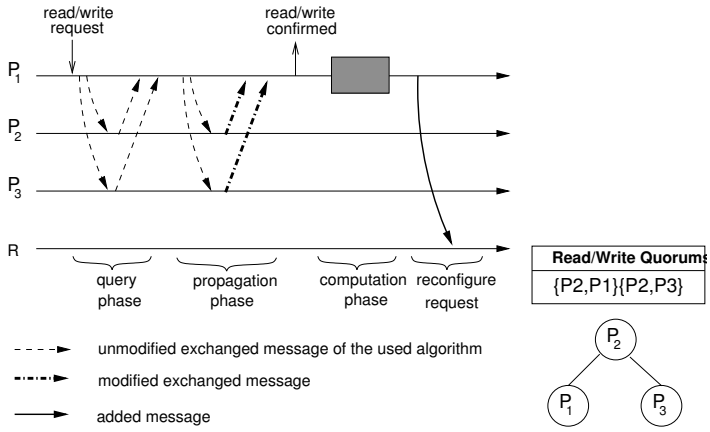


FIG. 3.2. *After an elementary permutation between P_a and P_b*

4. The Read/Write Algorithm. Our elementary permutation of the tree-structured coterie must be embedded in a suitable read/write algorithm. This algorithm must take care of concurrent accesses as well



(a) Read/Write operation sequence (b) 3 processors of a tree coterie

FIG. 4.1. *Quorum based Read/Write Atomic Service*

as dynamic reconfiguration of the tree-structured coterie with our permutation scheme. We present next an existing algorithm chosen from the literature of distributed systems. This algorithm is the one presented in [14].

4.1. The Used Atomic Read/Write Service. This algorithm is composed of two interfaces: the user interface that permits to access data by the well known read/write operations and the management interface which is used to reconfigure the current coterie.

User Interface. The read/write operations of a data item consist of two phases: a query phase and a propagation phase. At the time of the query phase, a read quorum is contacted and each processor of the quorum returns the value and the version of their local replica as well as the value and the version of their current coterie. Once all the answers are collected, the most recent version of the data is extracted. According to the operation, this recent version is either incremented and propagated (write) or simply propagated (read). In this propagation phase, the new value and the new version of the data is assigned to a write quorum. These two phases are carried out systematically for a read or write operation, that makes it possible to update the obsolete copies even when a data is read.

Figure 4.1 illustrates these two phases. Sub-figure *(a)* shows us the read/write protocol. Sub-figure (b) gives us the used three processors of a coterie and its corresponding read/write quorums. We emphasize that in the tree-structured coterie, read and write quorums are identical.

Management Interface. The service also possesses a management interface which makes it possible the dynamic reconfiguration of the used coterie. A reconfiguration can be carried out as the read or write operations are being performed. A reconfigurer is in charge of the reconfiguration process. It can be either an elected or a dedicated processor. The reconfiguration protocol is composed of three phases. During the installation phase, the reconfigurer contacts a minimal group of processors. The contacted group is the union of a read quorum and a write quorum to which the new configuration is send by the reconfigurer. The processors return to the reconfigurer the value and the version of their local replica. When all the answers are arrived, the reconfigurer enters the propagation phase. During this phase, a write quorum of this new coterie is contacted which guarantees the consistency among the replicas. Finally, the confirmation phase confirms the installation of the new configuration by sending it to a write quorum of this new coterie.

4.2. Extensions of the Atomic Read/Write Service. We propose to extend the atomic read/write service by adding three functionalities which are beyond the scope of [14] and that permit to realize our elementary permutation.

These functionalities are as follows:

1. **an information policy**: to gather information concerning the load of each processor,
2. **a selection policy**: to define the possible and convenient moment of reconfiguration,
3. **a reconfiguration policy**: to choose one of the previously defined permutations to apply if a reconfiguration can be carried out.

Next, we introduce the extensions which correspond to the elementary permutation.

4.2.1. Elementary Permutation Based Extension. Here we describe our elementary permutation based extension of the read/write atomic service. This extension consists of the following three policies.

The information policy. An elementary permutation can be carried out by having the load information of the processors that must be permuted. Hence, the major role of the information policy is to acquire, during the propagation phase of the read/write operations, the load of the processors of a quorum. The collected loads are enough to apply one or more elementary permutations within only one quorum: a path from the root to a leaf.

The selection policy. The choice of when to reconfigure is the major role of the selection policy. The question here is when to apply one or more elementary permutations. This choice is made naturally at the time of each operation, once the propagation phase is completed and the operation is confirmed. Each operation leads to contact a quorum. If this quorum contains a pattern where a parent is more loaded than a child then an elementary permutation can be carried out.

The reconfiguration policy. After each propagation phase, once the loads are known and the patterns are identified in the used read/write quorum, all possible elementary permutations can be applied. So after the reconfiguration, the path from the root to a leaf contains the processors in descending order of loads. The less loaded processor of the initial quorum is at the root and the more loaded one is at the leaf.

In Figure 4.1 the propagation phase's bold lines correspond to our information policy. Just after the propagation phase, the processor performs the reconfiguration policy by computing all the permutations that can be achieved. We call this phase a *computation phase*. If there exists one or several permutations to be applied, the new configuration is sent to the reconfigurer in order to perform the actual reconfiguration which is depicted as the *reconfiguration request*.

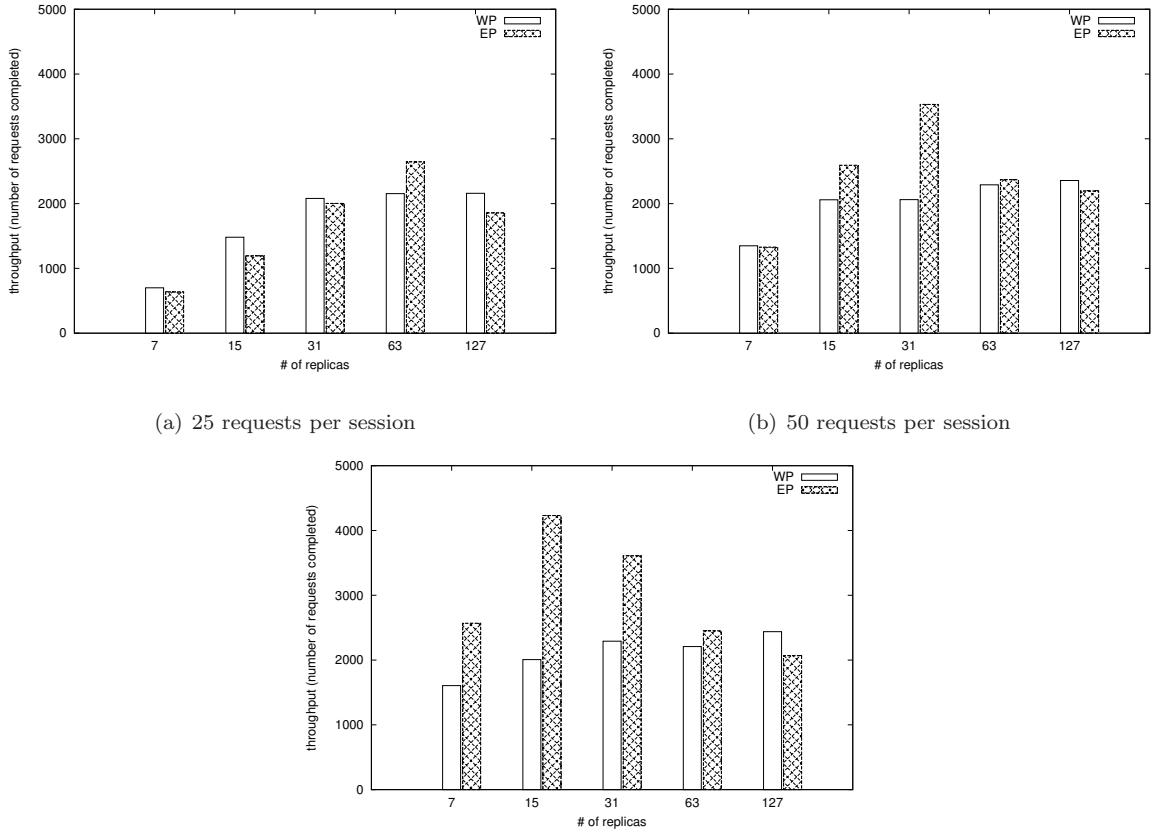
5. Performance Evaluation. In order to evaluate our algorithm, we implemented it in the Neko simulation environment [21]. We then proceeded to a simulation campaign where we studied several different characteristics such as throughput and scalability.

We realized each simulation by taking into consideration the following two cases: without reconfiguration (**WP**) and with elementary permutation (**EP**). For each case, we used different numbers of replicas: 7, 15, 31, 63 and 127, each corresponding to a number of processors. Each processor has its own load that can evolve randomly during the simulation. The time during which the load remains constant is called the session time. The session time follows the Poisson law that permits a long enough session. The number of read/write requests executed by each processor is also a parameter of the simulation. What we first found out is the fact that there is a strong relationship between the session time and the number of requests in our simulation results. So we took into account different number of requests per session to present our simulation results. The simulation time was fixed so that we can compare the number of confirmed requests of our different cases.

5.1. The Impact of the Number of Requests. Figure 5.1 represents the throughput as a function of the number of replicas. Each sub-figure corresponds to a specific number of requests per session. The first observation we can make is that the differences between **WP** and **EP** are more significative when the number of requests per session is high. At the beginning of a new session a well loaded coterie is naturally configured using one or several elementary permutations. The more requests occurs before the next session, the more important is the impact of the first permutation. To sum up, the ratio between the number of requests and the number of reconfigurations is higher when there are more requests per session. The results shown in Figure 5.1 illustrate that **EP** is better when there are up to 50 requests per session.

5.2. Low Scalability of the Elementary Permutation Algorithm. Even if there is a high number of requests: up to 50 requests, there are cases where **EP** have a lower throughput than **WP**. Figure 5.1(b) and 5.1(c), the case with 127 replicas performs better throughput without permutation than with elementary permutation. If there is a large number of replicas, there are too much reconfigurations and the benefits of using elementary permutations are lost.

5.3. Increased Throughput with Elementary Permutation. In the others cases, when there aren't a lots of replicas and with a high number of requests per session, **EP** permits to have better throughput than **WP**. In several cases, Figure 5.1(b) for 15,31 and 63 replicas and Figure 5.1(c) for 7,15,31 and 63 replicas, we notice an enhanced throughput which is between 20% and 50% with respect to a non permuted system.



(a) 25 requests per session

(b) 50 requests per session

(c) 100 requests per session

FIG. 5.1. The throughput as a function of the number of replicas. There are two different represented series: **WP** (Without Permutation) and **EP** (Elementary Permutation)

6. Conclusion and Future Work. We have linked the construction of a coterie to the loads of its processors. We have defined the notion of a quorum's load as well as coterie's load. These definitions helped us to propose a new reconfiguration protocol to apply to a tree-structured coterie: the principle of *elementary permutation*. We have shown that this permutation protocol permits to ameliorate the load of a coterie. We have extended an atomic read/write algorithm that permits dynamic reconfigurations of a coterie so that we can embed our permutation protocol. The simulation campaigns that we have carried out thanks to the Neko simulator, showed us the benefits of our simulations. The elementary permutation allows to improve the throughput by 50% for a small number of processors and a large number of requests per session. But it is subject to the scale. If there is an important number of nodes, do not apply any permutation seems to be better.

We claim that our algorithm can be used in a grid computing environment. It could be difficult to use this protocol in peer-to-peer grid environment because of the high number of replicas but the *elementary permutation* protocol can be used in collaborative computing if there is no need to have a large number of replicas. Elementary permutation protocol can also be used in data intensive read/write applications if the number of requests is high and if it is needed to maintain sequential consistency between data copies.

In this paper we show that the elementary permutation algorithm does not scale well. If the number of replicas is large, do not apply any permutation is better. One of our major concern is to find an algorithm that can resolve this scalability problem. Another issue that our algorithm does not address is the network delay. This is done in [20] where the authors map quorum onto a physical network with a fixed topology. However they left as an open problem to take into account the service time. We do the opposite and we didn't use the network delay in our load model. It would be interesting to construct coteries taking into account the network latency and the processor's load to have a more precise load model and to find a more suitable solution.

REFERENCES

[1] A. E. Abbadi, D. Skeen, and F. Cristian, *An efficient, fault-tolerant protocol for replicated data management*, in Proceedings of the fourth ACM SIGACT-SIGMOD symposium on Principles of database systems, ACM Press, 1985, pp. 215–229.

[2] D. Agrawal and A. E. Abbadi, *An efficient and fault-tolerant solution for distributed mutual exclusion*, ACM Transactions on Computer Systems, 9 (1991), pp. 1–20.

[3] ———, *Using reconfiguration for efficient management of replicated data*, IEEE Transactions on Knowledge and Data Engineering, 8 (1996), pp. 786–801.

[4] L. Alvisi, E. T. Pierce, D. Malkhi, M. K. Reiter, and R. N. Wright, *Dynamic byzantine quorum systems*, in DSN'00: Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8), Washington, DC, USA, 2000, IEEE Computer Society, p. 283.

[5] P. Barford and M. Crovella, *Critical path analysis of TCP transactions*, in ACM SIGCOMM Internet Measurement Workshop, June 2001.

[6] G. Cao and M. Singhal, *A delay-optimal quorum-based mutual exclusion algorithm for distributed systems*, IEEE Trans. Parallel Distrib. Syst., 12 (2001), pp. 1256–1268.

[7] P. Carmi, S. Dolev, S. Har-Peled, M. J. Katz, and M. Segal, *Geographic quorum systems approximations*, in to appear in algorithmica, December 2003.

[8] S. Cheung, M. Ammar, and A. Ahamad, *The grid protocol: A high performance scheme for maintaining replicated data*, IEEE Transactions on Knowledge and Data Engineering, 4 (1992), pp. 582–592.

[9] I. Foster, C. Kesselman, and S. Tueke, *The anatomy of the grid - enabling scalable virtual organizations*, Intl J. Supercomputer Applications, (2001).

[10] A. W. Fu, *Delay-optimal quorum consensus for distributed systems*, IEEE Transactions on Parallel and Distributed Systems, 8 (1997), pp. 59–69.

[11] H. Garcia-Molina and D. Barbara, *How to assign votes in a distributed system*, Journal of the ACM, 32 (1985), pp. 841–860.

[12] D. K. Gifford, *Weighted voting for replicated data*, in Proceedings of the seventh ACM symposium on Operating systems principles, ACM Press, 1979, pp. 150–162.

[13] A. Kumar, *Hierarchical quorum consensus: A new algorithm for managing replicated data*, IEEE Transactions on Computers, 40 (1991), pp. 996–1004.

[14] N. A. Lynch and A. A. Shvartsman, *Robust emulation of shared memory using dynamic quorum-acknowledged broadcasts*, in FTCS '97: Proceedings of the 27th International Symposium on Fault-Tolerant Computing (FTCS '97), IEEE Computer Society, 1997.

[15] D. Peleg and A. Wool, *Crumbling walls: a class of practical and efficient quorum systems*, Distributed Computing, 10 (1997), pp. 87–97.

[16] M. Rabinovich and E. D. Lazowska, *The dynamic tree protocol: avoiding "graceful degradation" in the tree protocol for distributed mutual exclusion*, in Conference Proceedings., Eleventh Annual International Phoenix Conference on Computers and Communications, Scottsdale, AZ, USA, April 1992, pp. 101–107.

[17] ———, *Improving fault tolerance and supporting partial writes in structured coterie protocols for replicated objects*, in Proceedings of the 1992 ACM SIGMOD international conference on Management of data, ACM Press, 1992, pp. 226–235.

[18] F. Thibolt, I. Frain, and A. M'Zoughi, *Virtualisation du stockage dans les grilles informatiques* , in 16me Rencontres Francophones du Paralllisme, (Renpar'05) , Croisic, France, ASF/ACM/Sigops, 6-8 avril 2005, pp. 219–224.

[19] R. Thomas, *A majority consensus approach to concurrency control for multiple copy databases*, ACM Transactions on Database Systems, 4 (1979), pp. 180–209.

[20] T. Tsuchiya, M. Yamaguchi, and T. Kikuno, *Minimizing the maximum delay for reaching consensus in quorum-based mutual exclusion schemes*, IEEE Transactions on parallel and distributed systems, 10 (1999), pp. 337–345.

[21] P. Urban, X. Dfago, and A. Schiper, *Neko: A single environment to simulate and prototype distributed algorithms*, in 15th Int'l Confereence on Information Networking (ICOIN-15), 2001, pp. 503–511.

Edited by: Dana Petcu
Received: May 31, 2007
Accepted: June 18, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 249–261. http://www.scpe.org
ISSN 1895-1767

A TOOL FOR A TWO-LEVEL DYNAMIC LOAD BALANCING STRATEGY IN SCIENTIFIC APPLICATIONS*

RICOLINDO L. CARIÑO† AND IOANA BANICESCU‡

Abstract. This paper describes a dynamic load balancing tool intended for computational investigators who have little familiarity with programming for a message-passing environment. Motivated by the PAR DOALL directive available in some compilers for shared-memory systems, the tool is designed to simplify the manual conversion of sequential programs containing computationally intensive one- or two-dimensional loops with independent iterates into parallel programs that execute with high efficiency on general-purpose clusters. The tool implements a dynamic loop scheduling strategy to address load imbalance which may be induced by the non-uniformity of loop iterate times, and by the heterogeneity of processors. The tool is based on the Message Passing Interface library for wide availability. Experimental results of two scientific applications that utilize the tool on a Linux cluster are presented to demonstrate sample achievable performance, and to underscore the effectiveness of the two-level dynamic load balancing strategy.

Key words. dynamic load balancing, loop scheduling, image denoising, vector functional coefficient autoregressive model

1. Introduction. Automatic parallelization of sequential programs containing computationally-intensive parallel loops is supported by many compilers for shared-memory environments. For example, on a Sun multiprocessor and with a Sun Fortran 77/90 compiler [38], simple insertion of the `C$PAR DOALL` directive before a parallel loop will instruct the compiler to generate parallel code. Similar directives are also provided in Sun C [37] and OpenMP [9]. A clause attached to the directive may specify Guided self scheduling [42] or factoring [26] as the loop scheduling technique for load balancing, or a fixed chunk size may be supplied. Other scheduling techniques have been developed, such as trapezoid self scheduling [46], weighted factoring [25], adaptive weighted factoring [5, 7, 11, 12], and, adaptive factoring [4, 6]; however, at the time writing, these techniques are not supported by the said clause.

In contrast, much manual effort is required to convert a serial program with a parallel loop so that it will execute on a message-passing platform. Code for explicitly assigning the loop iterates to processors have to be written, which can be a daunting task for those not familiar with developing message-passing programs. The code can statically assign iterates to the participating processors before runtime—a simple strategy that is appropriate if the iterates have uniform execution times and if the processors are homogeneous and start at the same time. Otherwise, chunks of iterates have to be dynamically assigned during runtime to processors that become idle. In this case, the chunk sizes must be chosen judiciously such that the processors will all finish at the earliest possible time. The loop scheduling techniques mentioned above may be utilized for computing the chunk sizes.

The contribution of this paper is a tool that is an "analogue" of the `C$PAR DOALL` directive for parallel loops in applications that will execute on message-passing clusters. The tool is designed to simplify the conversion of a sequential program into a message-passing parallel version with dynamic load balancing, but without extensive code modification. Essentially, one statement initializes the tool and five statements convert a parallel loop, leaving the code of loop body untouched. High performance is expected from the resulting parallel version because of the added dynamic load balancing capability to address both algorithmic and systemic sources of load imbalance. The target users of the tool are scientists who write code for novel computational techniques in a serial language, but require a cluster to extensively investigate the properties of the technique and to exercise the code. Examples of target users are statisticians who are developing new estimation procedures for which statistical properties require study through Monte Carlo simulations, and numerical analysts conducting accuracy studies or parametric studies for new numerical methods.

The rest of this paper is organized as follows. Section 2 identifies the class of applications which can take advantage of the proposed load balancing tool. The section also reviews dynamic loop scheduling techniques that have been utilized for determining processor loads. Section 3 describes the design of the tool and its

*This work was partially supported by the following National Science Foundation grants: ITR/DMS 0313274, EPS 0132618 and, CAREER 9984465.

†Center for Advanced Vehicular Systems – HPCC, PO Box 5405, Mississippi State University, MS 39762, rlc@cavs.msstate.edu

‡Department of Computer Science and Engineering, PO Box 9637, and Center for Computational Sciences – HPCC, PO Box 9627, Mississippi State University, MS 39762 ioana@cse.msstate.edu

interface to applications. Section 4 illustrates the use of the tool in two real applications: the simulation of a hybrid model for image denoising, and the investigation of the statistical properties of estimators and hypothesis tests related to the vector functional coefficient autoregressive (VFCAR) model for multivariate nonlinear time series. Section 5 gives a summary and describes future work.

2. Parallel loops and dynamic loop scheduling. The proposed dynamic load balancing tool is intended for serial applications that contain straightforward parallel loops (1D-loops) or nested parallel loops (2D-loops), as illustrated in Figure 2.1, using Fortran 90 notation.

```
a) 1D-loop
  ...
  DO I=1,N
    ...  I-ITERATE
  END DO
  ...
```

```
b) 2D-loop
    ...
  DO J=1,M  ! J-LOOP
     ...  PART A OF J-ITERATE
    DO I=1,N(J)  ! I-LOOP
     ...  I-ITERATE of J-ITERATE
    END DO ! i
    ...  PART B OF J-ITERATE
  END DO ! j
```

FIG. 2.1. *Target loops*

Parallel loops, or loops with no dependencies among their iterates, are major sources of concurrency in scientific applications. These are frequently targeted for parallelization to reduce application execution time, and therefore, to achieve high performance. The iterates of a parallel loop can be executed in any order or even simultaneously without affecting the correctness of the computations. However, minimizing the loop completion time is not straightforward. Factors such as the nonuniformity of iterate execution times, heterogeneity of the processors, and operating system interference during loop execution, give rise to unbalanced processor workloads, which ultimately lead to application performance degradation. These factors, arising from algorithmic and systemic irregularities that may not be known before the application starts, require the integration of dynamic load balancing into the strategy for parallel loop execution.

Load balancing during the execution of a parallel loop is achieved through *dynamic loop scheduling*, where chunks of loop iterates are executed concurrently by the participating processors. Many techniques for computing the chunk sizes have been proposed and implemented, including non-adaptive [32, 42, 26, 46, 25] and adaptive [5, 7, 4, 6, 11, 12] techniques, in addition to simple static chunking and self scheduling. The simple techniques and some of the non-adaptive techniques [42, 26] have been incorporated into compiler technologies for shared-memory environments, such as multiprocessors from Sun Microsystems [37, 38]. Thus, automatic parallelization could be achieved through judicious insertion of a compiler directive immediately before a parallel loop. To the authors' knowledge, this facility is not yet available for message-passing clusters, which typically have far more processors and interesting sources of irregularities than shared-memory multiprocessors.

A strategy for executing 2D-loops on shared-memory multiprocessors has been proposed and theoretically investigated [10, 24, 45]. This strategy describes a feedback-guided self scheduling (FGDLS) algorithm which uses a feedback mechanism to schedule a parallel loop within a sequentially executed outer loop. It has been shown to perform well for scheduling problems for which the load associated with the parallel loop changes relatively slowly as the outer loop executes sequentially. Sufficient conditions have been established for the convergence of the algorithm. The FGDLS differs significantly from the strategy underlying the tool proposed in this paper; here, the iterates of the outer loop are executed in parallel and the target environment is a message-passing architecture. Thus, at least two additional levels of complexity are addressed by the tool.

Some load balancing libraries have been developed for clusters [27, 34, 21], but these are for specific classes of applications that utilize standardized data structures. Previously, the authors have designed a load balancing tool which requires the user to provide a routine that encapsulates the computations of a loop iterate, and routines for migrating data between processors [13].

3. Tool design. A user of the tool proposed in this paper may be someone who has developed a new computational technique and has written a sequential program to investigate the properties of the technique or to exercise the code implementing the technique. The sequential program is essentially a computationally-

intensive parallel loop, and the user has realized that he will be able to conduct more extensive experiments and to publish results sooner if his investigations are carried out on a parallel system. Although automatic parallelization is supported by many compilers for shared-memory environments, the user prefers a message-passing cluster because of platform availability or because of the necessity for more processors. This is the ideal scenario in which the proposed dynamic load balancing can be utilized.

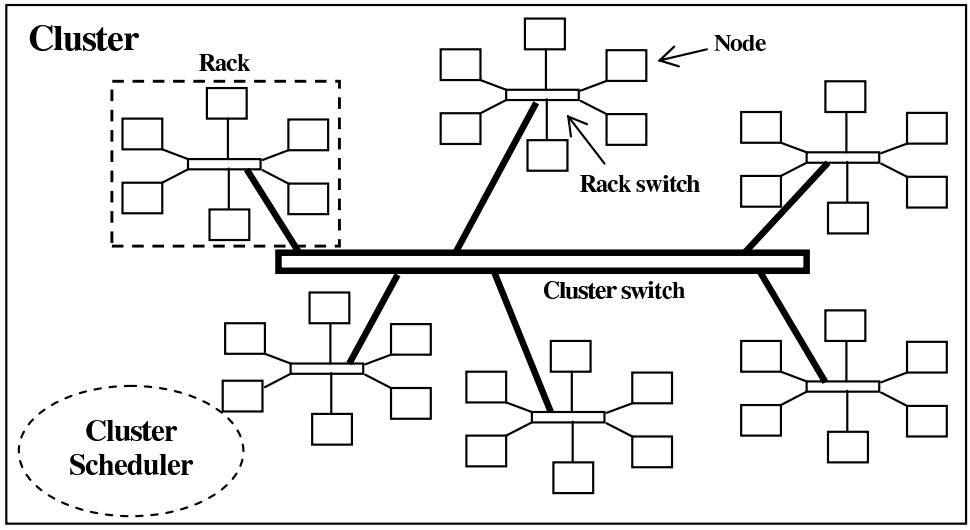


FIG. 3.1. *A popular interconnection network for clusters*

The tool simplifies the parallelization and load balancing of applications that contain computationally-intensive 1D- and 2D-loops (Figure 2.1) on message-passing clusters. These clusters are usually organized into racks that are connected by a cluster switch, each rack consisting of a number of nodes connected by a rack switch, and each node containing one or more processors. Figure 3.1 illustrates a popular interconnection configuration. Heterogeneity is inherent in such a cluster, more so if it was constructed incrementally over a period of time, because the processors would have different capabilities. Rates of communication between processors are also variable. Typically, the cluster scheduler attempts to assign nodes from a single rack to a job for efficient communications. Even with excellent job scheduling algorithms, the scattering of processors across a number of racks occurs with a high probability, especially for jobs that request large numbers of processors. Thus, applications running on clusters typically need to incorporate load balancing for highest possible performance.

Figure 3.2 illustrates the modification of the code of the 1D-loop to integrate the tool. Only a few lines are added: in summary, the original `DO` loop (capitalized) is converted to a `while` loop where chunks of iterates can be executed concurrently on different processors. Since the `I-ITERATE` invokes CPU-intensive computations, which may be expressed in hundreds or thousands of lines of code, the additional code to integrate the tool constitutes a tiny percentage of the total number of lines of code for the application.

The module `DLS` (abbreviation for Dynamic Loop Scheduling) contains the type definition of `infoDLS` and the codes for the `DLS_*` routines. Based on the Message-Passing Interface (MPI) library [22], the routines implement a scheduler–worker strategy of load balancing, where the scheduler participates in executing loads, in addition to being responsible for assigning loads.

The `DLS` routines used in Figure 3.2 are:

`DLS_Setup(MPI_COMM_WORLD,info)` initializes a dynamic loop scheduling environment (Figure 3.3) on `MPI_COMM_WORLD`. Information about this environment is maintained in the data structure `info`.

`DLS_StartLoop(info,1,N,method)` is the synchronization point for the start of loop execution. `(1,N)` is the loop range, and `method` is a user-specified index for the selected loop scheduling technique. The following techniques are implemented: static scheduling, a modification of fixed size chunking [32], guided self scheduling [42], factoring [26], variants of adaptive weighted factoring [5, 7, 11, 12], and adaptive factoring [4, 6].

`DLS_Terminated(info)` returns true if all loop iterates have been executed.

```
program Application_with_1D_loop
  use DLS
  include 'mpif.h'
  type (infoDLS) info
  integer method, iStart, iSize, iIters
  double precision iTime
  integer mpierr
  call MPI_Init (mpierr)
  ...
  method = ...
  call DLS_Setup (MPI_COMM_WORLD, info)
  call DLS_StartLoop (info, 1, N, method)
  do while ( .not. DLS_Terminated(info) )
    call DLS_StartChunk(info,iStart,iSize)
    DO I=iStart, iStart+iSize-1
      ...  I-ITERATE
    END DO
    call DLS_EndChunk (info)
  end do
  call DLS_EndLoop (info, iIters, iTime)
  ...
```

FIG. 3.2. *Dynamic load balancing of an application containing a 1D-loop.*

`DLS_StartChunk(info,iStart,iSize)` returns a range for a chunk of iterates. This range starts with iterate `iStart` and contains `iSize` iterates.
`DLS_EndChunk(info)` signals the end of execution of a chunk of iterates. Internally, a worker processor requests its next chunk from the scheduler.
`DLS_EndLoop(info,iIters,iTime)` is the synchronization point at the end loop execution. `iIters` is the number of iterates done by the calling processor, and `iTime` is the cost (in seconds) measured using MPI_Wtime(). `iIters` and `iTime` are useful for assessing the performance gains achieved by dynamic load balancing. For example, the sum of the `iTime`s from all participating processors gives an estimate of the cost of executing the loop on a single processor.

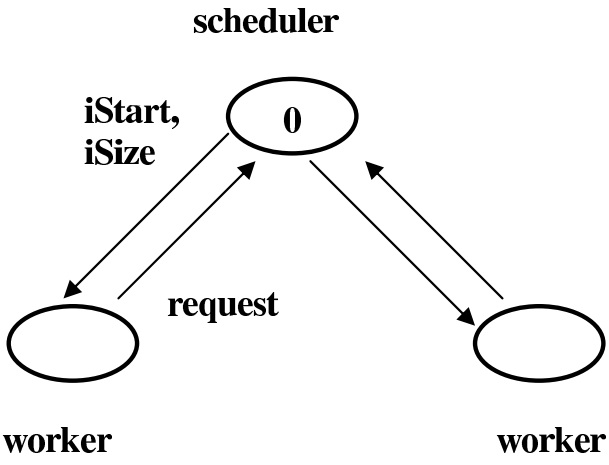


FIG. 3.3. *Scheduler-worker strategy for dynamic loop scheduling*

After loop execution, the results of the computations (in `I-ITERATE`) will be distributed among the participating processors. A reduction operation like MPI_Reduce() may be necessary to collect the results in one processor, or MPI_Allreduce to make the results available to all processors in `MPI_COMM_WORLD`. This would be the responsibility of the user, since DLS only manipulates the indices of the loop. Information about the mapping of the chunks of iterates to processors is maintained in the `chunkMap` component of the `infoDLS` structure.

Figure 3.4 illustrates the integration of DLS into an application containing a 2D-loop. (The original code is capitalized.) An iterate of the outer loop may be composed of a series of `I-LOOP`s; the code for each one goes through the same modification.

```
program Application_with_2D_loop
  use DLS
  include 'mpif.h'
  ...
  type (infoDLS) iInfo, jInfo
  integer iMethod, iStart, iSize
  integer jMethod, jStart, jSize
  integer iIters, jIters
  double precision iTime, jTime
  integer coordinator, minSize, maxSize
  ...
  iMethod = (loop scheduling technique)
  jMethod = (loop scheduling technique)
  minSize = (min. processors for inner loop)
  maxSize = (max. processors for inner loop)
  coordinator = 0
  call DLS_GroupSetup (MPI_COMM_WORLD,&
    coordinator,minSize,maxSize,jInfo,iInfo)
  call DLS_StartLoop (jInfo, 1, M, jMethod)
  do while ( .not. DLS_Terminated(jInfo) )
    call DLS_StartChunk (jInfo, jStart, jSize)
    DO J=jStart, jStart+jSjze-1  ! J-LOOP
      ...  PART A OF J-ITERATE
      call DLS_StartLoop (iInfo,1,N(J),iMethod)
      do while ( .not. DLS_Terminated(iInfo) )
        call DLS_StartChunk(iInfo,iStart,iSize)
        DO I=iStart, iStart+iSize-1  ! I-LOOP
          ...  I-ITERATE of J-ITERATE
        END DO ! i
        call DLS_EndChunk (iInfo)
      end do ! while
      call DLS_EndLoop (iInfo, iIters, iTime)
      ...  PART B OF J-ITERATE
    END DO ! j
    call DLS_EndChunk (jInfo)
  end do ! while
  call DLS_EndLoop (jInfo, jIters, jTime)
  ...
```

FIG. 3.4. *Parallelization and load balancing of a 2D-loop.*

The environment to execute a 2D-loop is initialized by `DLS_GroupSetup()`, which splits `MPI_COMM_WORLD` into a number of non-overlapping communicators `iComm`s and one communicator `jComm` (see Figure 3.5). The processors residing in the same rack are combined into an `iComm` for efficient communications, subject to the constraint that the size of `iComm` is in the range [`minSize, maxSize`], to match the amount of concurrency in the inner loop. Processors from different racks may be combined in an `iComm` to satisfy `minSize`, or processors from a single rack may be split into many `iComm`s to satisfy `maxSize`. The `jComm` is comprised of the `coordinator` and the rank-0 processors of the `iComm`s. Information regarding this two-level setup is stored in the data structures `jInfo` and `iInfo`. Except for the `coordinator`, all processors participate in executing iterates. The iterates of the `J-LOOP` are dynamically scheduled in `jComm`, following the scheduler-worker strategy depicted by Figure 3.3. Chunks of `J-ITERATE`s are concurrent executed on the `iComm`s, each `iComm` also following the same scheduler-

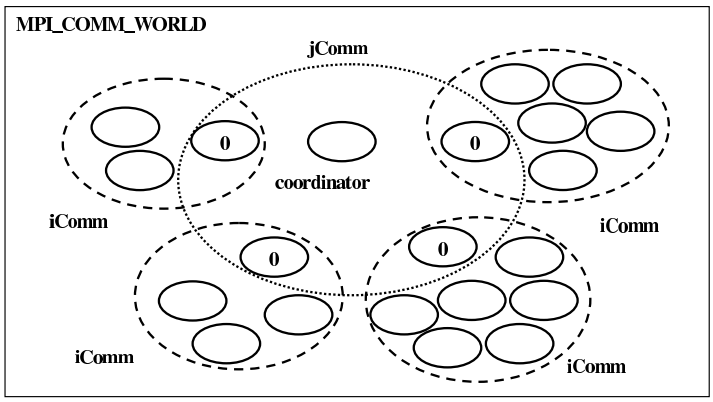


FIG. 3.5. *Two-level setup for 2D-loops. Dynamic load balancing occurs simultaneously on all the communicators.*

worker strategy as the `jComm`. In fact, the same `DLS` routines are used, but with different arguments, notably, `iInfo` in an `iComm` and `jInfo` in the `jComm`.

Within an `iComm`, the processors execute a chunk of `J-ITERATE`s sequentially, in lock step. The processors redundantly compute `PART A` and `PART B` of a `J-ITERATE`. However, the processors cooperatively execute the `N(J)` iterates the `I-LOOP`. For this strategy to be applicable, `M` must be significantly larger than the number of `iComm`s, and that each `N(J)` must be significantly larger than the number of processors in an `iComm`. Otherwise, the number of chunks to be scheduled may not be sufficient in order to achieve good load balancing.

4. Applications. The dynamic load balancing tool has been integrated into a number of scientific applications. To illustrate the utility and the performance improvement achievable by the tool, this section presents timing results for the simulation of an image denoising model, and the simulation of a vector functional coefficient autoregressive (VFCAR) model for multivariate nonlinear time series.

These applications were executed on the heterogeneous general-purpose EMPIRE cluster of the Mississippi State University. The cluster can be abstracted as in Figure 3.1 and has a total of 1038 processors. A rack contains 32 nodes of dual 1.0GHz or 1.266GHz Pentium III processors and 1.25GB RAM. Each node is connected to a 100Mb/s Ethernet rack switch. The rack switches are connected by a gigabit Ethernet cluster switch. Installed software includes RedHat Linux and PBS. The general submission queue allows 64-processor, 48-hour jobs; a special queue allows 128-processor, 96-hour jobs from a restricted set of users. According to the Top 500 Supercomputer Sites list published in June 2002, EMPIRE then was the 126th fastest computer in the world and the 10th among educational institutions in the U.S.

4.1. Image denoising. Denoising is an important image processing (IP) step for various image-related applications and often necessary as a pre-processing for other imaging tasks such as segmentation and compression. Thus, image denoising methods have occupied an important position in IP, computer graphics, and their applications. Recently, as the field of IP requires higher levels of reliability and efficiency, various powerful tools of partial differential equations (PDEs) and functional analysis have been successfully applied to image restoration [1, 15, 19, 28, 35, 39, 41, 43, 50] and color processing [8, 20, 29, 31, 44]. In particular, a considerable amount of research has been carried out for the theoretical and computational understanding of the total variation (TV) model [43] and its variants [1, 15, 16, 19, 28, 30, 35, 36, 47].

However, most of those denoising models may lose fine structures of the image due to a certain undesired dissipation. As remedies, the employment of the G-norm [36] and iterative refinement [40] have been studied. But these new methods are either difficult to minimize utilizing the Euler-Lagrange equation approach or have the tendency to keep an observable amount of noise. Recently, in order to overcome the drawbacks, one of the authors suggested the method of nonflat time evolution (MONTE) [18] and the equalized net diffusion (END) approach [17]. The MONTE and END techniques are applicable to various (conventional) denoising models as either a time-stepping procedure or a variant of mathematical modeling.

As another remedy to the undesired dissipation, fourth-order PDE models have emerged [23, 33, 49]. In particular, the Laplacian mean-curvature (LMC) model has been paid a particular attention due to its potential capability to preserve edges of linear curvatures. However, it has been numerically verified [48] that the LMC

model can easily introduce granule-shaped spots to restored images. To overcome the granularity, a hybrid model which combines a TV-based model and the LMC has been proposed [14], briefly described below.

The Laplacian mean-curvature (LMC) model is:

$$\frac{\partial u}{\partial t} + \Delta\kappa(u) = \beta(f-u), \tag{4.1}$$

where $\beta \geq 0$, a constraint coefficient, and $\kappa(u)$ denotes the mean-curvature defined as

$$\kappa(u) = \nabla \cdot \Big(\frac{\nabla u}{|\nabla u|}\Big).$$

In equation (4.1), f is a contaminated image and the solution u represents a denoised image. The LMC model has a major drawback: granularity. The restored image can easily incorporate granule-shaped spots. The LMC model also shows staircasing, a phenomenon that tends to make the restored image locally constant. However, it is relatively easy to cure [29, 35].

As a remedy for the granule-shaped spots introduced by the LMC model, consider the following hybrid model

$$\frac{\partial u}{\partial t} - \sigma\,\widetilde{\kappa}(u) + \Delta\widetilde{\kappa}(u) = \beta(f-u), \tag{4.2}$$

where $\sigma \geq 0$ is a regularization parameter and

$$\widetilde{\kappa}(u) = |\nabla u|\,\kappa(u) = |\nabla u|\,\nabla \cdot \Big(\frac{\nabla u}{|\nabla u|}\Big).$$

Here the gradient magnitude $|\nabla u|$ has been incorporated into $\widetilde{\kappa}(u)$, as a scaling factor, in order to reduce staircasing [35]. The second-order term is introduced for 4.2 to hold a certain degree of maximum principle, with which the model in turn can eliminate the granularity [48].

Equation 4.2 is a *generalized LMC* (GLMC) model with the three model parameters (β, σ and α). The iterations to solve the differential equations involve two extra algorithm parameters: Δt and n. Thus, for a given contaminated image, values have to be selected for these parameters which result in the best restored image. However, when the original uncontaminated image is not known, assessing the quality of the restored image is difficult, if not impossible. In order to gain insight on the influence of these parameters on the quality of the restored image, the model is simulated on known images with synthetically-added Gaussian noise. As a measure of the quality of the restored images, the peak signal-to-noise ratio (PSNR) is adopted. The PSNR is defined as

$$\mathrm{PSNR} \equiv 10\log_{10}\left(\frac{\sum_{ij} 255^2}{\sum_{ij}(g_{ij}-u_{ij})^2}\right)\mathrm{dB},$$

where g denotes the original uncontaminated image and u is the restored image.

Establish uncontaminated image g
Add Gaussian noise to g to produce contaminated image f
Establish parameter counts $\mathrm{N}_\beta, \mathrm{N}_\sigma, \mathrm{N}_\alpha, \mathrm{N}_{\Delta t}, \mathrm{N}_n$
Establish parameter values $\beta[1], \ldots, \beta[\mathrm{N}_\beta]$; $\sigma[1], \ldots, \sigma[\mathrm{N}_\sigma]$;
$\alpha[1], \ldots, \alpha[\mathrm{N}_\alpha]$; $\Delta t[1], \ldots, \Delta t[\mathrm{N}_{\Delta t}]$; $n[1], \ldots, n[\mathrm{N}_n]$
For each combination of $\beta, \sigma, \alpha, \Delta t, n$ values
  Apply denoising procedure on f to produce restored image u
  Calculate PSNR; output $\beta, \sigma, \alpha, \Delta t, n$ and PSNR
End for

FIG. 4.1. *Outline of parametric study for image denoising model*

The parametric study to investigate the influence of the parameters $\beta, \sigma, \alpha, \Delta t$ and n on PSNR is outlined by the the pseudo-code in Figure 4.1. Various plots from the outputs of the study could be produced, including

animations of PSNR as a function of β, σ, α, with either Δt or n fixed and using the other as the variable for the animation.

The number of combinations of parameter values is simply $N_\beta \times N_\sigma \times N_\alpha \times N_{\Delta t} \times N_n$, which could be huge even for small to moderate values of the parameter counts. Fortunately, the denoising procedure can be computed simultaneously for several combinations of the parameters, on a parallel machine. However, the denoising procedure performs nonuniform amounts of computations for each parameter combination; therefore, dynamic load balancing is necessary for efficient utilization of the parallel machine. These characteristics render the parameter study an ideal test application for the load balancing tool described in Section 3. The next two figures summarize the performance of the resulting parallel code for the parameter study with $N_\beta = 9$, $N_\sigma = 9$, $N_\alpha = 9$, $N_{\Delta t} = 9$ and $N_n = 15$, for a total of 98,415 parameter combinations.

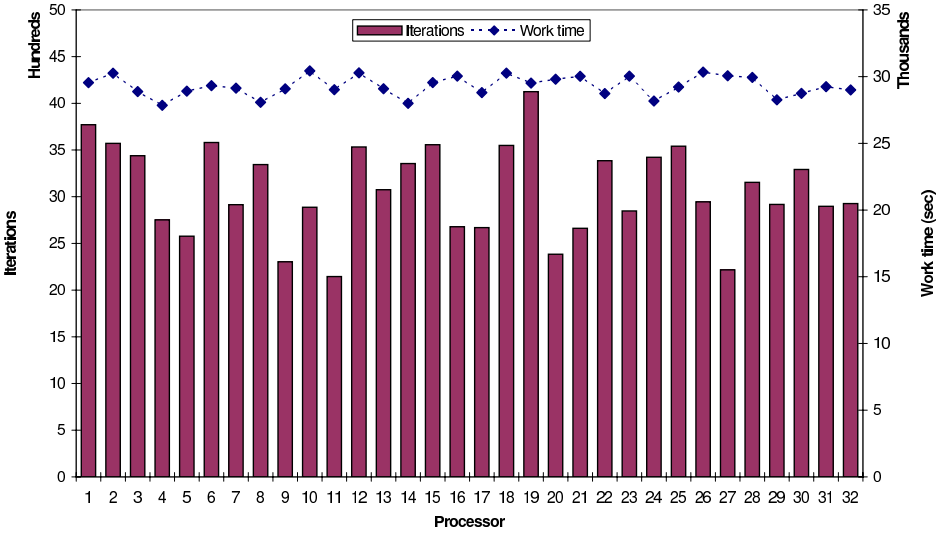


FIG. 4.2. *Distribution of iterations and work times for the parametric study on the image* LenaGray256

Figure 4.2 gives a summary of the performance of the parallel code for the parametric study on the image *LenaGray256*. This study was submitted as a 32-processor job on the EMPIRE cluster; the cluster scheduler assigned homogeneous processors to the job. Since jobs were also executing on the cluster along with the study, the contention for network resources was a source of system-induced load imbalance. However, the major source of load imbalance was the nonuniform amount of computations required by the denoising procedure for different sets of parameter values. The left axis (for the bars) denotes the number of iterations of the loop in Figure 4.1 executed by a processor, while the right axis (for the diamonds) denotes the time in seconds taken by the processor to execute the iterations. The large differences in the number of iterations done by the processors is evidence for application-induced load imbalance. However, the difference between the maximum and minimum work times is only 2581.3 seconds, which is a relatively narrow range. The job time measured by the cluster scheduler was 8.453 hours. An estimate of the sequential cost of the study is 260.4547 hours (∼10.9 days), which is the sum of the work times of the 32 processors. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (260.4547)/(32×8.453) = 0.963. The high efficiency indicates that the dynamic load balancing tool successfully addressed the load imbalance.

Figure 4.3 gives the summary for the parametric study on the image *BlackCircle*. The cluster scheduler again assigned homogeneous processors to the study, and the job time was 39.546 hours. The differences in iteration counts are significant, indicating the presence of application-induced load imbalance. An estimate of the sequential cost is 1,223.279 hours (∼51 days), which is the sum of the work times of the 32 processors. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (1223.279)/(32×39.546) = 0.967.

4.2. Vector nonlinear time series. A vector time series is a set of observations of multiple related phenomena across time. The mathematical underpinnings of the statistical analysis of time series incorporate the correlation across time and between series—properties that complicate statistical theory. This is especially true for nonlinear models, where mathematical theory may be extremely difficult, even intractable. Although

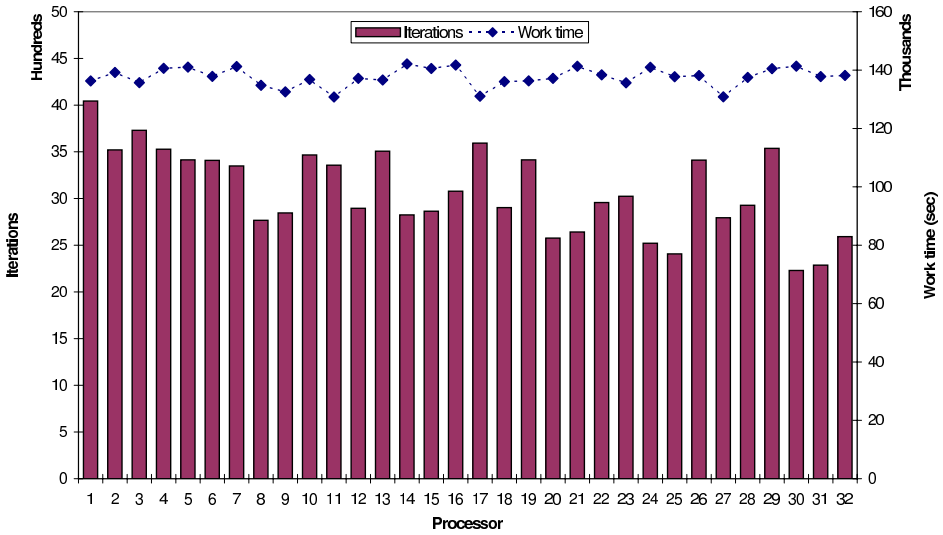


FIG. 4.3. *Distribution of iterations and work times for the parametric study on the image* BlackCircle

complicated in both presentation and theory, vector nonlinear time series are especially useful for describing complicated nonlinear dynamic structures that exists in many time-dependent multivariate series.

Let $\boldsymbol{Y}_t = (Y_{1,t}, \ldots, Y_{k,t})'$ denote the vector time series at time $t = 1, 2, \ldots, T$. Then the vector functional coefficient autoregressive model of order p (VFCAR(p)) is defined as

$$\boldsymbol{Y}_t = \boldsymbol{f}^{(0)}(\boldsymbol{Z}_t) + \sum_{j=1}^{p} \boldsymbol{f}^{(j)}(\boldsymbol{Z}_t)\boldsymbol{Y}_{t-j} + \boldsymbol{\varepsilon}_t, \quad t = p+1, \ldots, T, \tag{4.3}$$

where $\boldsymbol{f}^{(j)},\ j = 0, \ldots, p$ are $k \times k$ matrices whose elements are real-valued measurable functions that change as a function of the (possible vector-valued) $\boldsymbol{Z}_t$, and which have continuous second-order derivatives. The error terms $\boldsymbol{\varepsilon}_t$ in (4.3) are such that for each i, the series $\{\varepsilon_{i,t}\}_{t=1}^{T}$ is a white noise sequence, independent of $\{\boldsymbol{Y}_t\}_{t=1}^{T}$. However contemporaneous cross-correlation may exist between $\{\varepsilon_{i,t}\}$ and $\{\varepsilon_{j,t}\},\ i \neq j$. The primary motivation for studying this model is that specific choices for the elements of the $\boldsymbol{f}^{(j)}$ yield parametric models.

The VFCAR(p) model may be considered a hybrid of non-parametric and parametric models since the autoregressive structure is assumed, but there is little or no information about the form of the elements of the $\boldsymbol{f}^{(j)}$. As such, estimation of the parameters of the VFCAR(p) model is done nonparametrically via local regression. Simultaneous estimation of the elements of the $\boldsymbol{f}^{(j)}$ provides improved statistical efficiency when the error terms have positive cross-correlation. In the process of fitting the model (4.3), modified multifold cross-validation is used to determine an optimal bandwidth and value for p by finding the pair of values that minimize the accumulated prediction error. This multistage procedure requires an immense number of arithmetic operations on a univariate series. That number increases exponentially for multivariate series.

The mathematical complexity of the statistical procedures in using the VFCAR model are highly complicated, necessitating the use of simulation to study their properties. Consequently, Monte Carlo simulation is often relied upon to give direction, to interpret, and to explain complex analytical results. In general, it can be said that as the number of Monte Carlo replications increases, the result of the simulation approaches the "truth". The more complex the structure, the larger the number of replications needs to be.

The examination of statistical properties of methods related to the VFCAR models via simulations on a single processor would require execution times of a few weeks or even months. Statisticians have been known to base empirical results on a relatively small number of simulation replications, sacrificing precision, accuracy, and possibly compromising the reliability of results in the interest of time. As a result, techniques which may require thousands of replications for accuracy and reliability often have at most, a few hundred. Fortunately, the replications are amenable for computation in parallel. Thus, parallel processing technology can be exploited to enable the extensive simulation of a variety of models within reasonable running time limits.

```
Input model specifications; no_reps, nh, no_bs_reps
...
DO rep_no=1,no_reps  ! replications
  ...
  DO i1=1,nh+1       ! bandwidth
    ...
  END DO
  ...
  DO i2=1,no_bs_reps ! bootstrap test
    ...
  END DO
  ...
END DO
...
```

FIG. 4.4. *Outline of VFCAR simulation*

Figure 4.4 gives a high-level outline of the simulation procedure to investigate the statistical properties of estimators and hypothesis tests related to the VFCAR model for multivariate nonlinear time series. Results of computational experiments in which the simulation procedure is treated as a 1D-loop (i. e., only the replication loop is parallelized) are reported in [3, 2]. Estimated efficiencies of up to 97% were achieved on 64 processors.

As a further demonstration of the capability of the dynamic load balancing tool, the simulation was treated as a 2D-loop. Therefore, a parallelization strategy similar to Figure 3.4 was followed. The 2D-loop version of the application, with `no_reps`=10000, `nh`=50, and `no_bs_reps`=500, was submitted as a 64-processor job to the EMPIRE cluster. The adaptive factoring technique [4, 6] for loop scheduling was specified. Owing to the small amount of concurrency in the inner (bandwidth and bootstrap test) loops, `maxSize` was set to 4. In the sample run summarized by Figure 4.5, the cluster scheduler assigned to the job 2, 10, 10 and 42 processors from racks 4, 5, 8 and 11, respectively. Rack 11 contains 1.266GHz processors, while the other racks contain 1.0GHz processors. Eighteen (18) `iComm`s were formed: `iComm`s 2–6 and 12–15, each with 4×1.266GHz processors; `iComm` 1 with 3×1.266GHz processors; `iComm` 16 with 2×1.266GHz processors; `iComm`s 7–10, each with 4×1.0GHz processors; `iComm`s 11, 17, and 18, each with 2×1.0GHz processors; and the remaining processor as the `coordinator`.

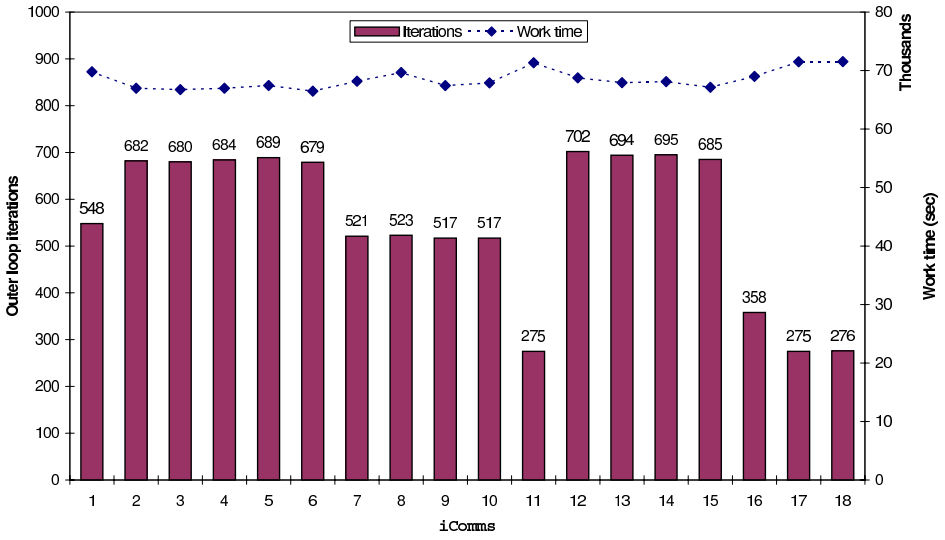


FIG. 4.5. *Distribution of outer loop iterations and work times for VFCAR simulation*

Figure 4.5 illustrates the number of replications (outer loop iterates) executed by each `iComm` and their corresponding work times. The heterogeneity of the `iComm`s is reflected in unequal numbers of replications

executed. But the variation in the work times is small: the average of the absolute deviation (AAD) of the work times is only 2% of the mean work time, computed as follows: if x_i is the work time of the ith `iComm` and $\bar{x}$ is the mean work time, then AAD=$\sum_{i=1}^{18} |x_i - \bar{x}|/(18 * \bar{x})$. The parallel job time measured by the cluster scheduler is 19.89 hours. An estimate of the sequential cost is 1193.06 hours (∼49.71 days), obtained as the sum of the costs from each `iComm`. Thus, an estimate of the efficiency is: (estimated sequential cost)/(parallel cost) = (1193.06)/(64*19.89) = 0.937.

Note that the estimated efficiency of the 2D-loop version (93.7%) of the simulation is slightly less than that of the 1D-loop version (97%). This is to be expected since the overhead of the 2D-loop version is higher. In any case, the results are indicative of the effectiveness of the two-level dynamic load balancing strategy implemented by the tool.

5. Summary and Future Work. An interesting load balancing problem arises when running a scientific application containing CPU-intensive one-dimensional or two-dimensional parallel loops on a general-purpose cluster: the loop iterates may have nonuniform execution times, and the processors allocated to the application may be heterogeneous and may reside on different cluster racks. These sources of load imbalance may not be known or predictable before the application starts.

This paper describes a dynamic loop scheduling tool to address the above problem. The tool is especially designed for computational investigators who have little familiarity with developing message-passing programs. The tool easily integrates into an existing sequential application, after minor code modifications, to produce a message-passing version. For a 2D-loop, the tool follows a two-level strategy, where at one level, chunks of iterates of the outer loop are executed concurrently, and simultaneously at another level, chunks of iterates of the inner loop are also executed concurrently.

The tool has been integrated into nontrivial sequential applications. The applications mentioned in this paper achieved estimated efficiencies in excess of 90% on a general-purpose heterogeneous cluster. The tool is being integrated into other scientific applications; discoveries made through these applications will be reported elsewhere. The use of the tool for dynamic scheduling of 3D-loops (or even higher) is being investigated. Intuitively, a 3D-loop can be implemented using the same two-level strategy, where a 1D-loop is executed at one level, while the 2D-loop is executed at the other level. Results of these investigations will be reported in the future.

Acknowledgments. We thank Hyeona Lim, Neil Williams and Seongjai Kim for their collaboration on the simulation of a hybrid model for image denoising. We also thank Jane Harvill and John Lestrade for their collaboration on the analysis of gamma-ray burst datasets using vector functional coefficient autoregressive time series models.

REFERENCES

[1] L. Alvarez, P. Lions, and M. Morel, *Image selective smoothing and edge detection by nonlinear diffusion. II*, SIAM J. Numer. Anal., 29 (1992), pp. 845–866.

[2] I. Banicescu, R. L. Cariño, J. L. Harvill, and J. P. Lestrade, *Computational challenges in vector functional coefficient autoregressive models*, in Lecture Notes in Computer Science 3514, Computational Science—ICCS 2005, V. S. et al., ed., Springer-Verlag, 2005, pp. 237–244.

[3] ———, *Simulation of vector nonlinear time series on clusters.*, in Proceedings of the 19th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2005, p. CDROM.

[4] I. Banicescu and Z. Liu, *Adaptive factoring: A dynamic scheduling method tuned to the rate of weight changes*, in Proceedings of the High Performance Computing Symposium (HPC) 2000, 2000, pp. 122–129.

[5] I. Banicescu and V. Velusamy, *Performance of scheduling scientific applications with adaptive weighted factoring*, in Proceedings of the 15th IEEE International Parallel and Distributed Processing Symposium—10th Heterogenous Computing Workshop (IPDPS-HCW 2001) CDROM, IEEE Computer Society, Apr. 2001.

[6] ———, *Load balancing highly irregular computations with the adaptive factoring*, in Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium—11th Heterogeneous Computing Workshop (IPDPS-HCW 2002) CDROM, IEEE Computer Society, 2002.

[7] I. Banicescu, V. Velusamy, and J. Devaprasad, *On the scalability of dynamic scheduling scientific applications with adaptive weighted factoring*, Cluster Computing: The Journal of Networks, Software Tools and Applications, 6 (2003), pp. 215–226.

[8] P. Blomgren and T. Chan, *Color TV: Total variation methods for restoration of vector valued images*, IEEE Trans. Image Process., 7 (1998), pp. 304–309.

[9] O. A. R. Board, *OpenMP specifications*, 2003. `http://www.openmp.org/specs`

[10] J. M. Bull, *Feedback guided dynamic loop scheduling: Algorithms and experiments*, in Lecture Notes in Computer Science

1740, Proceedings of the 4th International Euro-Par Conference (EuroPar'98), D. Pritchard and J. Reeve, eds., Springer-Verlag, 1998, pp. 377–382.

[11] R. L. Cariño and I. Banicescu, *Dynamic scheduling parallel loops with variable iterate execution times*, in Proceedings of the 16th IEEE International Parallel and Distributed Processing Symposium - 3rd Workshop on Parallel and Distributed Scientific and Engineering Computing With Applications (IPDPS-PDSECA 2002) CDROM, IEEE Computer Society, 2002.

[12] ———, *Load balancing parallel loops on message-passing systems*, in Proceedings of the 14th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2004), S. Akl and T. Gonzales, eds., ACTA Press, 2002, pp. 362–367.

[13] ———, *A load balancing tool for distributed parallel loops*, Cluster Computing, 8 (2005), pp. 313–321.

[14] R. L. Cariño, I. Banicescu, H. Lim, N. Williams, and S. Kim, *Simulation of a hybrid model for image denoising*, in Proceedings of the 20th International Parallel and Distributed Processing Symposium, IEEE Computer Society, 2006, p. (to appear).

[15] F. Catte, P. Lions, M. Morel, and T. Coll, *Image selective smoothing and edge detection by nonlinear diffusion.*, SIAM J. Numer. Anal., 29 (1992), pp. 182–193.

[16] Y. Cha and S. Kim, *Edge-forming methods for color image zooming.* (accepted to IEEE Trans. Image Process.).

[17] ———, *Equalized net diffusion (END) for the preservation of fine structures in PDE-based image restoration.* (submitted to IEEE Trans. Image Process.).

[18] ———, *MONTE: The method of nonflat time evolution in PDE-based image restoration.* (submitted to IEEE Trans. Image Process.).

[19] T. Chan, S. Osher, and J. Shen, *The digital TV filter and nonlinear denoising*, Technical Report #99-34, Department of Mathematics, University of California, Los Angeles, CA 90095-1555, October 1999.

[20] T. Chan and J. Shen, *Variational restoration of non-flat image features: Models and algorithms*, SIAM J. Appl. Math., 61 (2000), pp. 1338–1361.

[21] K. Devine, B. Hendrickson, E. Boman, M. St. John, and C. Vaughan, *Zoltan: A Dynamic Load Balancing Library for Parallel Applications; User's Guide*, Sandia National Laboratories, Albuquerque, NM, 1999. Tech. Report SAND99-1377 `http://www.cs.sandia.gov/Zoltan/ug_html/ug.html`

[22] M. Forum, *The Message Passing Interface Standard*, 1995. `http://www-unix.mcs.anl.gov/mpi`

[23] J. Greer and A. Bertozzi, *Traveling wave solutions of fourth order pdes for image processing*, SIAM J. Numer. Anal., 36 (2004), pp. 38–68.

[24] D. Hancock, R. Ford, T. Freeman, and J. Bull, *An investigation of feedback guided dynamic scheduling of nested loops*, in Proceedings of the 2000 International Conference on Parallel Processing Workshops (ICPPW'00), IEEE Computer Society, 2000, pp. 315–321.

[25] S. F. Hummel, J. Schmidt, R. N. Uma, and J. Wein, *Load-sharing in heterogeneous systems via weighted factoring*, in Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures (SPAA 1996), 1996, pp. 318–328.

[26] S. F. Hummel, E. Schonberg, and L. E. Flynn, *Factoring: A method for scheduling parallel loops*, Communications of the ACM, 35 (1992), pp. 90–101.

[27] G. Karypis and V. Kumar, *ParMETIS: Parallel graph partitioning and sparse matrix ordering.* `http://www-users.cs.umn.edu/~karypis/metis/parmetis/index.html` 2003.

[28] S. Kim, *Edge-preserving noise removal: Motion by mean curvature.* (submitted to SIAM J. Sci. Comput.).

[29] ———, *PDE-based image restoration: A hybrid model and color image denoising.* (accepted to IEEE Trans. Image Process.).

[30] S. Kim and H. Lim, *A non-convex diffusion model for simultaneous image denoising and edge enhancement.* (submitted to Electron. J. Differ. Equ.).

[31] R. Kimmel and N. Sochen, *Orientation diffusion or how to comb a porcupine?*, Special issue on PDEs in Image Processing, Computer Vision, and Computer Graphics, J. Visual Comm. Image Representation, 13 (2002), pp. 238–248.

[32] C. Kruskal and A. Weiss, *Allocating independent subtasks on parallel processors*, IEEE Transactions on Software Engineering, SE-11 (1985), pp. 1001–1016.

[33] M. Lysaker, A. Lundervold, and X. Tai, *Noise removal using fourth-order partial differential equations with applications to medical magnetic resonance images in space and time*, IEEE Trans. Image Process., 12 (2003), pp. 1579–1590.

[34] B. Maerten, D. Roose, A. Basermann, J. Fingberg, and G. Lonsdale, *DRAMA: A library for parallel dynamic load balancing of finite element applications*, in Euro-Par '99: Proceedings of the 5th International Euro-Par Conference on Parallel Processing, London, UK, 1999, Springer-Verlag, pp. 313–316.

[35] A. Marquina and S. Osher, *Explicit algorithms for a new time dependent model based on level set motion for nonlinear deblurring and noise removal*, SIAM J. Sci. Comput., 22 (2000), pp. 387–405.

[36] Y. Meyer, *Oscillating Patterns in Image Processing and Nonlinear Evolution Equations*, vol. 22 of University Lecture Series, American Mathematical Society, Providence, Rhode Island, 2001.

[37] S. Microsystems, *Forte C 6 /Sun Workshop 6 Compilers C User's Guide*, 2003. `http://docs.sun.com/source/806-3567/parallel.html`

[38] ———, *Fortran User's Guide*, 2003. `http://docs.sun.com/source/806-3591/E_directives.html`

[39] M. Nitzberg and T. Shiota, *Nonlinear image filtering with edge and corner enhancement*, IEEE Trans. on Pattern Anal. Mach. Intell., 14 (1992), pp. 826–833.

[40] S. Osher, M. Burger, D. Goldfarb, J. Xu, and W. Yin, *Using geometry and iterated refinement for inverse problems (1): Total variation based image restoration*, CAM Report #04-13, Department of Mathematics, UCLA, LA, CA 90095, 2004.

[41] P. Perona and J. Malik, *Scale-space and edge detection using anisotropic diffusion*, IEEE Trans. on Pattern Anal. Mach. Intell., 12 (1990), pp. 629–639.

[42] C. Polychronopoulos and D. Kuck, *Guided self-scheduling: A practical scheduling scheme for parallel supercomputers*, IEEE Transactions on Computers, C-36 (1987), pp. 1425–1439.

[43] L. Rudin, S. Osher, and E. Fatemi, *Nonlinear total variation based noise removal algorithms*, Physica D, 60 (1992), pp. 259–268.
[44] N. Sochen, R. Kimmel, and R. Malladi, *A general framework for low level vision*, IEEE Trans. Image Process., 7 (1998), pp. 310–318.
[45] T. Tabirca, L. Freeman, S. Tabirca, and L. T. Yang, *Feedback guided dynamic loop scheduling; a theoretical approach*, in Proceedings of the 2001 International Conference on Parallel Processing Workshops (ICPPW'01), IEEE Computer Society, 2001, pp. 115–121.
[46] T. H. Tzen and L. M. Ni, *Trapezoid self-scheduling: A practical scheduling scheme for parallel computers*, IEEE Transactions on Parallel Distributed Systems, 4 (1993), pp. 87–98.
[47] L. Vese and S. Osher, *Numerical methods for p-harmonic flows and applications to image processing*, Technical Report #01-22, Department of Mathematics, University of California, Los Angeles, CA 90095, USA, August 2001. (To appear in SIAM Numer. Anal.).
[48] N. Williams, H. Lim, and S. Kim, *Numerical schemes for the Laplacian mean-curvature flow and their applications to image denoising*. (In preparation).
[49] Y. You and M. Kaveh, *Fourth-order partial differential equations for noise removal*, IEEE Trans. Image Process., 9 (2000), pp. 1723–1730.
[50] Y. You, W. Xu, A. Tannenbaum, and M. Kaveh, *Behavioral analysis of anisotropic diffusion in image processing*, IEEE Trans. Image Process., 5 (1996), pp. 1539–1553.

Edited by: Dana Petcu
Received: May 29, 2007
Accepted: June 18, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 263–270. http://www.scpe.org
ISSN 1895-1767

ROBUST PARALLEL IMPLEMENTATION OF A LANCZOS-BASED ALGORITHM FOR AN STRUCTURED ELECTROMAGNETIC EIGENVALUE PROBLEM

MIGUEL O. BERNABEU*, MÁRIAM TARONCHER†, VÍCTOR M. GARCÍA‡, AND ANA VIDAL§

Abstract. This paper describes a parallel implementation of a Lanczos-based method to solve generalised eigenvalue problems related to the modal computation of arbitrarily shaped waveguides. This efficient implementation is intended for execution mainly in moderate-low cost workstations (2 to 4 processors). The problem under study has several features: the involved matrices are sparse with a certain structure, and all the eigenvalues needed are contained in a given interval. The novel parallel algorithms proposed show excellent speed-up for small number of processors.

Key words. large eigenvalue problem, structured matrices, microwaves

1. Introduction and examples. This paper is focused on the parallelisation of a Lanczos-based method for the solution of the following generalised eigenvalue problem: Given a symmetric pencil $\mathbf{A}x = \lambda\mathbf{B}x$, find all the generalised eigenvalues (and the corresponding eigenvectors) comprised in a given interval. This interval contains a large number of eigenvalues.

An efficient sequential method was already proposed in [1]. However, when the number of desired eigenvalues is very large, the execution time is still too long. A first parallel algorithm was recently introduced in [2], using MPI and a distributed-memory approach. The results presented in that paper show that the method parallelises extremely well.

A code based in the proposed technique will be included in a CAD tool for design of passive waveguide components. However, this CAD tool will usually run in low cost workstations or, at most, small PC clusters. For these small systems, a different approach should be chosen.

Therefore, the main goal of this paper is to explore different parallel programming approaches for the implementation of the sequential technique described in [1], in low cost workstations and small clusters.

Three different approaches have been examined: First, we have designed an OpenMP version of the Lanczos algorithm to take advantage of two-processor machines. Next, we implemented a version for distributed memory machines using MPI (Message Passing Interface), to execute it on clusters of PCs. Finally, a mixed approach was proposed in order to achieve optimum performance on clusters of two-processors.

A number of modifications have been carried out lately in the algorithm, to improve the reliability of the code, these shall be described as well. The main corrections have been i) the inclusion of ARPACK [7] routines for the extraction of all the generalised eigenvalues in a small subinterval, ii) the correction of the algorithm for balancing workload, and iii) the improvement of the linear solver, formerly based in the LU-Schur complement and now based on the LDL_t decomposition.

The paper is organised as follows: first, we will briefly outline the sequential problem (described in [1]), including the algorithmic modifications. Then, the new parallelisation schemes will be completely described, taking into account the different proposed options: i. e. MPI, OpenMP, MPI+OpenMP and so on. Finally, some numerical results are shown, and then the conclusions of this work are given.

2. Problem Description and Sequential Algorithm.

2.1. The electromagnetic problem. In this study, the efficient and accurate modal computation of arbitrary waveguides is based on the Boundary Integral - Resonant Mode Expansion (BI-RME) method (see the detailed formulation in [1, 3]). This technique provides the modal cut-off frequencies of an arbitrary waveguide from the solution of two eigenvalue problems. The first one is a generalised eigenvalue problem that models the transversal electric (TE) family of modes of the arbitrary waveguide. The structure of the corresponding matrices $\mathbf{A}$ and $\mathbf{B}$, shown in Fig. 2.1, presents a very sparse nature that is conveniently exploited in this work.

*Dpt. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, mbernabeu@dsic.upv.es

† Dpt. de Comunicaciones, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, matacal@iteam.upv.es

‡Dpt. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, vmgarcia@dsic.upv.es

§Dpt. de Comunicaciones, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022 Valencia, Spain, avidal@dcom.upv.es

Both matrices have a non-zero main diagonal, and a small $N \times N$ block in the right, bottom corner. Furthermore, the B matrix has two thin nonzero stripes $\mathbf{R}$ (with dimensions $N \times M$) and $\mathbf{R_t}$ ($M \times N$), in the last N rows and the last N columns. The size of the matrices is $(M+N) \times (M+N)$, but since M is far larger than N the matrices are very sparse (see [1]). This situation is given when a large number of cut-off frequencies is demanded. The transversal magnetic (TM) family of modes can be also formulated as a generalised eigenvalue problem (see [1]) with matrices $\mathbf{A}$ and $\mathbf{B}$ very similar to those explained before for the TE modes.

Here we will consider only the TE case.

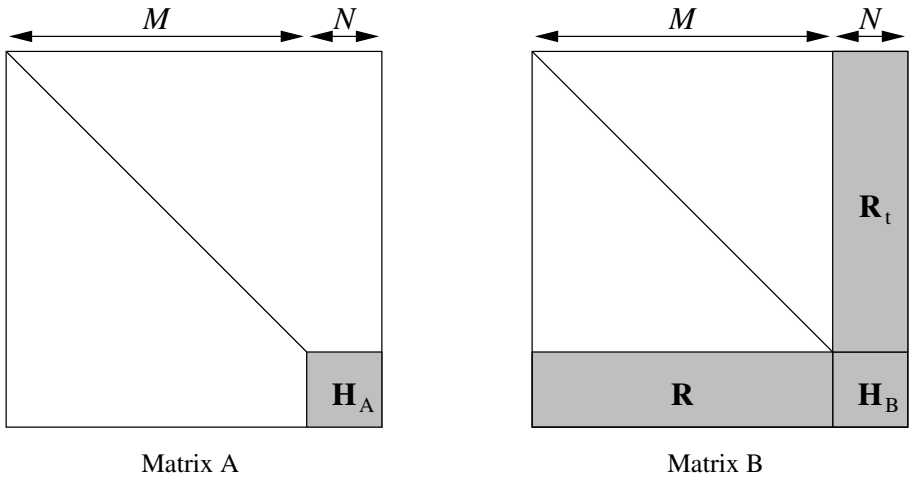


FIG. 2.1. *Structured matrices* $\mathbf{A}$ *and* $\mathbf{B}$ *for the TE problem in a ridge waveguide.*

2.2. The sequential algorithm.

2.2.1. Shift-and-Invert Lanczos' algorithm. The standard techniques for generalised eigenvalue problems is the QZ algorithm. However, as was described in [1], in this case is not efficient since it does not use the structure of the matrices.

The technique proposed in [1] by the authors is based on Lanczos algorithm [6]. This algorithm, in its most basic form, allows the computation of a reduced number of extremal eigenvalues (the largest or smallest in magnitude). However, given a real number (usually called *shift*) σ, Lanczos' algorithm can be applied to the matrix $\mathbf{W} = (\mathbf{A} - \sigma\mathbf{B})^{-1}\mathbf{B}$. Lanczos' algorithm applied to this matrix will deliver as result the eigenvalues of the original problem closer to the shift σ. (This is called the "Shift-and-Invert" version of the Lanczos' algorithm.) The application of the Lanczos' method to this problem requires the solution of several linear systems, with $\mathbf{A} - \sigma\mathbf{B}$ as coefficient matrix. However, the structure of the matrices $\mathbf{A}$ and $\mathbf{B}$ allows a very efficient solution of these systems, using the Schur complement method. This method, described in [1] for this problem, was based in the LU decomposition; one of the algorithmic improvements mentioned above has been to change the LU-based technique to a LDL_t based algorithm, described next.

2.2.2. LDL_t **decomposition.** Let us now find out how is the $\mathbf{LDL_t}$ decomposition of the matrices in our case. For a matrix $(\mathbf{A} - \sigma\mathbf{B})$ with $\mathbf{A}$ and $\mathbf{B}$ as above, we can write

$$\begin{aligned}\mathbf{A} - \sigma\mathbf{B} &= \begin{pmatrix} \mathbf{U}_\sigma & \mathbf{R}_{\sigma t} \\ \mathbf{R}_\sigma & \mathbf{H}_\sigma \end{pmatrix} = \begin{pmatrix} \mathbf{D} & \mathbf{0} \\ \mathbf{F} & \mathbf{T} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D}_l & \mathbf{0} \\ \mathbf{0} & \mathbf{D}_s \end{pmatrix} \cdot \begin{pmatrix} \mathbf{D_t} & \mathbf{F_t} \\ \mathbf{0} & \mathbf{T_t} \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{D} \cdot \mathbf{D}_l \cdot \mathbf{D_t} & \mathbf{D} \cdot \mathbf{D}_l \cdot \mathbf{F_t} \\ \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{D}_t & \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{F_t} + \mathbf{T} \cdot \mathbf{D}_s \cdot \mathbf{T_t} \end{pmatrix} \end{aligned} \tag{2.1}$$

where the structure of matrix $\mathbf{A} - \sigma\mathbf{B}$ is identical to that of matrix $\mathbf{B}$ (Figure 2.1).

It is easy to check that we can take $\mathbf{D}$ as the identity matrix (since $\mathbf{U}_\sigma$ is diagonal), so that equating parts of this equation we arrive to the following procedure to compute the $\mathbf{LDL_t}$ decomposition:

1. Take $\mathbf{D}_l$ equal to $\mathbf{U}_\sigma$.
2. $\mathbf{F} = \mathbf{R}_\sigma \cdot \mathbf{D}_l^{-1}$ (trivial, since $\mathbf{D}_l$ is diagonal).
3. $\mathbf{T}$ and $\mathbf{D}_s$ are obtained computing the $\mathbf{LDL_t}$ decomposition of $\mathbf{H}_\sigma - \mathbf{F} \cdot \mathbf{D}_l \cdot \mathbf{F_t}$, through the LAPACK routine *dsytrf* .

2.2.3. Main interval decomposition. As we have mentioned before, the shift-and-invert version of the Lanczos' algorithm computes a subset of the spectrum centred in the shift point. The number of eigenvalues required will determine the number of iterations of the Lanczos' algorithm and its spatial cost [7]. Obviously, we cannot apply the Lanczos' algorithm to the main interval $[\alpha, \beta]$ where all the desired eigenvalues lie. The original problem should be split into many smaller ones to ensure the optimal performance of the Lanczos' algorithm.

As shown in [1], it is possible to use the Inertia Theorem to know in advance how many eigenvalues contain a given interval $[\alpha, \beta]$. For such interval, the $\mathbf{LDL_t}$ decompositions of $\mathbf{A} - \alpha\mathbf{B}$ (equal to $\mathbf{L}^\alpha\mathbf{D}^\alpha{\mathbf{L_t}}^\alpha$) and $\mathbf{A} - \beta\mathbf{B}$ (equal to $\mathbf{L}^\beta\mathbf{D}^\beta{\mathbf{L_t}}^\beta$) can be computed with a moderated cost (as described above). Then, the number of eigenvalues in the interval is simply the number $\nu(\mathbf{D}^\beta) - \nu(\mathbf{D}^\alpha)$, where $\nu(\mathbf{D})$ denotes the number of negative elements in the diagonal $\mathbf{D}$. It must be taken into account that the diagonal returned by *dsytrf* may not be completely "diagonal"; instead, it can be diagonal with $1 * 1$ and $2 * 2$ blocks, as a consequence of the special pivoting strategy. In this case, the eigenvalues of this special diagonal matrix can be easily found (solving the characteristic equation for the 2*2 blocks), which allows to compute the inertia quite efficiently anyway.

Thus, we can divide the original $[\alpha, \beta]$ interval into m subintervals $[\alpha_i, \beta_i]$ of different length, but containing nearly the same number of eigenvalues, and where the number of eigenvalues in each subinterval is known exactly. Therefore, the CPU time needed to compute the eigenvalues of every subinterval is expected to be nearly constant.

2.2.4. Sequential algorithm. The full sequential algorithm is as follows: The interval where lie the desired eigenvalues, $[\alpha, \beta]$, is divided in many small subintervals. Then, in each subinterval, a shift (possibly the middle point) is selected, and then the "Shift-and Invert" Lanczos algorithm is applied independently to each subinterval. This will compute all the eigenvalues in each subinterval, independently of the other subintervals. The number of subintervals and its width are chosen so that the number of eigenvalues on each subinterval is not too large.

This allows to obtain all the eigenvalues in the full interval in a reasonable time and without memory problems (see [1] for all the details).

ALGORITHM 1. **Sequential overall algorithm**.

INPUT: matrices $\mathbf{A}$ *and* $\mathbf{B}$*, the main subinterval* $[\alpha, \beta]$ *and the maximum number of eigenvalues per subinterval*
OUTPUT: eigenvalues of the pair $(\mathbf{A}, \mathbf{B})$ *contained in* $[\alpha, \beta]$ *ant its corresponding eigenvectors*

1. *Apply the Inertia Theorem to the full interval* $[\alpha, \beta]$ *to divide it into m smaller subintervals* $[\alpha_i, \beta_i]$

2. *for every subinterval* $[\alpha_i, \beta_i]$
4. $\sigma = (\beta_i - \alpha_i)/2$
5. *Apply Lanczos' shift-and-invert method to extract the eigenvalues closer to* σ *and its eigenvectors*
6. *end for*

7. *return eigenvalues and eigenvectors*

In the latest versions of the code, the robustness has been improved by using the ARPACK [7] routines for the symmetric generalised eigenvalue problem *dsaupd*. This routine is faster and safer than our previous versions of the Lanczos algorithm.

3. Parallel implementations.

3.1. Algorithmic approach. Clearly, the basic idea for the parallel implementation is to distribute the subintervals among the available processors; in each subinterval, the extraction of the eigenvalues will still be carried out in a sequential way.

Once we have computed the bounds of every $[\alpha_i, \beta_i]$ subinterval, m/p subintervals are assigned to each processor. This assignation is performed at the beginning of the algorithm, and there is no more communication among the processors until all of them have ended its work and results have been gathered.

As we have mentioned in Section 2.2.3, the CPU time needed to extract the eigenvalues of every subinterval is expected to be nearly constant. Thus, just distributing them among the available processors the work load balance is expected to be close to the optimal.

ALGORITHM 2. **Parallel overall algorithm**.

INPUT: matrices $\mathbf{A}$ *and* $\mathbf{B}$*, the main interval* $[\alpha, \beta]$ *and the maximum number of eigenvalues per subinterval*
OUTPUT: eigenvalues of the pair $(\mathbf{A}, \mathbf{B})$ *contained in* $[\alpha, \beta]$ *ant its corresponding eigenvectors*

Let us suppose that m *is multiple of* p

1. At the master processor
2. Apply the Inertia Theorem to the full interval $[\alpha, \beta]$ *to divide it into* m *smaller subintervals* $[\alpha_i, \beta_i]$
3. Assign m/p *subintervals to each processor*
4. End at the master processor

5. For each processor
6. for every assigned subinterval $[\alpha_i, \beta_i]$
7. $\sigma = (\beta_i - \alpha_i)/2$
8. Apply Lanczos' shift-and-invert method to extract the eig eigenvalues closer to σ *and its eigenvectors*
9. end for

10. Send eigenvalues and eigenvectors to the master processor
11. End for each processor

12. At the master processor
13. Gather the results from all the processors
14. End at the master processor

3.2. Implementation details. The new proposed algorithms have been implemented in Fortran 90, making use of the Intel Fortran Compiler for Linux. OpenMP and MPI standards have been used for the shared-memory version and distributed-memory version, respectively. In addition, BLAS and LAPACK [8] have been used whenever it was possible.

We have implemented three versions of Algorithm 2:

1. MPI version of algorithm 2
2. OpenMP version of algorithm 2
3. MPI+OpenMP version of algorithm 2

In the MPI version, all the processes read the input data from disk (matrices and main interval). Then, the main interval is divided with the technique described in the previous section. Next, a distributed algorithm is executed to assign the subintervals that should be solved by each process. Once it is done, every process solves its corresponding subintervals sequentially. Then, the results are gathered by the master process. This version is oriented to distributed memory machines, although it should work as well in shared memory machine.

In the OpenMP version, only the main thread reads the input data from disk. Then, the $[\alpha, \beta]$ interval is divided by the main thread, again. Next, the subintervals are assigned and distributed among the threads. This version is designed to run on shared memory machines.

Finally, the MPI+OpenMP version combines both techniques. In the first level of parallelism, a set of p MPI processes are spawned and they execute the MPI algorithm described before. Then, in the step where each MPI process solves its m/p subintervals, a second level of parallelism is introduced. Instead of sequentially solving those intervals, a group of p' OpenMP threads are created and the m/p intervals are divided among them in the same way described in the OpenMP version. There are $p * p'$ processors working on the solution of the problem. Note that this version is a combination of the two previous ones , and has been designed to run on a cluster of SMP machines.

4. Experimental results.

4.1. Description of the test environment. In order to test the performance of the three implemented versions of algorithm 2, we have chosen two different environments: an SMP Cluster and an SGI Computation Server.

The SMP Cluster consists of two Intel Xeon bi-processors running at 2.2 GHz with 4GB of RAM each one, interconnected through a Gigabit-Ethernet network.

The SGI Computation Server is an SGI Altix 3700. This machine is a cluster of 44 Itanium II tetraprocessors, although it has been designed as a ccNUMA machine [5] and therefore can be programmed as a SMP machine.

As mentioned previously, the algorithms have been designed to be included into a CAD tool of complex passive waveguide components. This kind of tools are expected to run in moderate-low cost workstations, so the SMP Cluster is the perfect testing environment. Despite of this, we have also chosen a more complex and powerful machine, the SGI Server, in order to test the algorithm performance using more expensive machines. Obviously, we will only use 4 of the 44 processors available for fair comparison purposes with the cheaper machine.

4.2. Experimental results. The following tables show the execution times of the implementations listed in section 3 for both test environments.

For the testbed, we have considered a single ridge waveguide described in [1].

TABLE 4.1
Execution time (s) for MPI implementation at the SMP Cluster.

$M+N$	$p=1$	$p=2$	$p=4$
5000	71.68	40.92	20.45
8000	199.26	121.22	67.98
11000	426.32	257.13	140.06
14000	772.10	413.06	221.21
17000	1247.71	655.40	367.26
20000	1685.27	1003.56	540.88

TABLE 4.2
Execution time (s) for OpenMP implementation at the SMP Cluster.

$M+N$	$p=1$	$p=2$
5000	71.68	38.11
8000	199.26	109.78
11000	426.32	246.32
14000	772.10	419.12
17000	1247.71	646.51
20000	1685.27	963.91

4.3. Analysis of the experimental results. The previous results show that the method described in section 2 parallelises extremely well in affordable machines. The key points for this good behaviour are the

TABLE 4.3
Execution time (s) for MPI+OpenMP implementation at the SMP Cluster.

$M+N$	$p=1$	$p=4$
5000	71.68	20.53
8000	199.26	61.59
11000	426.32	134.88
14000	772.10	216.84
17000	1247.71	333.86
20000	1685.27	534.69

TABLE 4.4
Execution time (s) for OpenMP implementation at the SGI Server.

$M+N$	$p=1$	$p=2$	$p=3$	$p=4$
5000	44.14	25.44	18.66	14.95
8000	161.99	86.46	69.25	55.67
11000	321.68	185.16	148.37	133.35
14000	598.13	337.35	249.26	247.38
17000	893.64	494.42	405.15	351.61
20000	1259.16	665.58	556.76	532.72

TABLE 4.5
Execution time (s) for MPI implementation at the SGI Server.

$M+N$	$p=1$	$p=2$	$p=3$	$p=4$
5000	44.14	23.69	16.17	13.09
8000	161.99	86.34	60.85	49.24
11000	321.68	172.88	117.61	91.53
14000	598.13	310.42	217.38	170.07
17000	893.64	498.16	304.24	241.64
20000	1259.16	658.08	446.46	349.44

application of Inertia Theorem to ensure a good work-load balance, as well as the absence of communications during the execution of the algorithm.

The different versions of the developed algorithms differ in the parallel programming standard used: MPI, OpenMP, or both of them. Both standards offer good performance and the final choice depends more on the machine architecture rather than on the sequential algorithm characteristics.

TABLE 4.6
Speed-up @ the SMP Cluster. Comparative study between OpenMP and MPI versions ($p=2$).

$M+N$	OpenMP	MPI
5000	1.88	1.75
8000	1.82	1.64
11000	1.73	1.66
14000	1.84	1.87
17000	1.93	1.90
20000	1.75	1.68

Table 4.6 shows the speed-up of MPI and OpenMP versions in a two-processor board (one of the nodes of the SMP Cluster). OpenMP results are slightly better than MPI ones. This result was expected because OpenMP can take more advantage of the shared memory architecture of the machine.

Table 4.7 shows the speed-up of MPI and MPI+OpenMP versions in a cluster of 2 two-processor boards. In this kind of environments with two levels of parallelism (shared memory at each node and distributed memory

TABLE 4.7
Speed-up @ the SMP Cluster. Comparative study between MPI+OpenMP and MPI versions ($p = 4$).

$M + N$	MPI+OpenMP	MPI
5000	3.49	3.51
8000	3.24	2.93
11000	3.16	3.04
14000	3.56	3.49
17000	3.74	3.40
20000	3.15	3.12

for the global view of the machine) the combination of MPI and OpenMP standards show better results than the use of MPI only. Again, OpenMP is taking advantage of shared memory features of the machine while MPI is not doing so.

TABLE 4.8
Comparative analysis between OpenMP and MPI versions @ SGI Server ($M + N = 20000$).

version	$p = 2$	$p = 3$	$p = 4$
MPI version	1,91	2,82	3,60
OpenMP version	1,89	2,26	2,36

Table 4.8 shows the speed-up of MPI and OpenMP versions at the SGI Server. In this machine, the MPI version scales better than OpenMP version. This rather surprising result is due to the scheduling policy. When the batch system runs the parallel algorithm, it can schedule the p threads/processes to different boards. With the MPI algorithm this does not create problems, since each process owns all the necessary data to perform its part of the algorithm. However, for the OpenMP implementation it is different, because all the threads need to access master thread's memory. This would create accesses to memory placed in a different board, which shall slow down the algorithm. Obviously, the problem worsens as the number of threads increases.

5. Conclusions. Three parallel implementations of a Lanczos-based method for solving a generalised eigenvalue problem have been successfully developed. The problem has got several distinct characteristics: matrices are sparse and structured, and the search of eigenvalues is reduced to a fixed interval.

The proposed technique parallelises very well and any of the implementations present very good speed-up even for a small number of processors.

OpenMP is the best choice for parallel programming of two-processors boards (and any SMP environment). For NUMA systems, it is concluded that OpenMP may present some problems and its use should be studied carefully.

Multi level programming (MPI + OpenMP) is the best choice for hybrid machines (those with two levels of parallelism), since this paradigm can take advantage of both shared and distributed memory features of the machine.

Finally, we can conclude that execution times in both machines are not too different, while speed-up is clearly better at the SMP Cluster. So, in this case, the performance-cost ratio is clearly better for the SMP Cluster.

Acknowledgement. Contract/grant sponsor: partially supported by Ministerio de Educación y Ciencia, Spanish Government, and FEDER funds, European Commission; contract/grant number: TIC2003-08238- C02-02, and by Programa de Incentivo a la Investigacion UPV-Valencia 2005 Project 005522

REFERENCES

[1] GARCÍA V. M., VIDAL A., BORIA V. E., VIDAL A. M.: *Efficient and accurate waveguide mode computation using BI-RME and Lanczos method,* Int. Journal for Numerical Methods in Engineering. DOI:10.1002/nme.1520 (2005).

[2] VIDAL A. M., VIDAL A., BORIA V. E., GARCÍA V. M.: *Parallel computation of arbitrarily shaped waveguide modes using BI-RME and Lanczos method,* Submitted to Int. Journal for Numerical Methods in Engineering. (2006).

[3] CONCIAURO G., BRESSAN M., ZUFFADA, C.: *Waveguide modes via an integral equation leading to a linear matrix eigenvalue problem,* IEEE Transactions on Microwave Theory and Techniques. (1984).

[4] Snir M., Otto S., Huss-Lederman S., Walker D. and Dongarra J.: MPI: The Complete Reference; MIT Press, (1996).
[5] Grbic A.: Assessment of Cache Coherence Protocols in Shared-memory Multiprocessors; Phd Thesis, University of Toronto (2003).
[6] A. Ruhe.: *Generalized Hermitian Eigenvalue Problem; Lanczos Method,* In Templates for the Solution of Algebraic Eigenvalue Problems: a Practical Guide. SIAM, Philadelphia, first edition, (2000).
[7] R. B. Lehoucq, D. C. Sorensen, and C. Yang: *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods* SIAM, Philadelphia, (1998).
[8] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen: *LAPACK Users' Guide.* SIAM, Philadelphia, (1999).

Edited by: Dana Petcu
Received: May 31, 2007
Accepted: June 18, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 271–280. http://www.scpe.org
ISSN 1895-1767

RWAPI OVER INFINIBAND: DESIGN AND PERFORMANCE

OUISSEM BEN FREDJ AND ÉRIC RENAULT*

Abstract. This paper presents the design of InfiniWrite, the implementation of a lightweight communication interface called RWAPI over the InfiniBand interconnect for clusters of PCs. Since the specifications of the InfiniBand interconnect provide many ways to transfer data, we are discussing some issues regarding the choices between InfiniBand capabilities. We implemented RWAPI on top of the grid-oriented architecture called GRWA and evaluated the communication performance. GRWA has been developed to provide performance to higher applications on a wide variety of architectures. We obtain a very low latency and a throughput very close to the maximum user bandwidth for messages as small as 4 kilo-bytes.

Key words. one-sided communications, cluster and grid computing, high-speed networks, performance.

1. Introduction. In recent years, clusters of workstations have become a viable alternative to supercomputer for high performance computing. The success of cluster computing has lead people to interconnect several clusters to form a powerful heterogeneous parallel machine called a cluster of clusters which is now widely used in the industry. High-speed network interconnects that offer low latency and high bandwidth have been one of the main reasons attributed to the success of cluster systems. Some of the leading high-speed networking interconnects include Gigabit-Ethernet, InfiniBand [19], Myrinet [9] and Quadrics [8]. Two common features shared by these interconnects are User-level networking and Direct Memory Access (DMA). The class of communication protocol that best uses these new features is the one-sided protocol. In this class, the completion of a send (resp. receive) operation does not require the intervention of the receiver (resp. sender) process to take a complementary action. RDMA should be used to copy data to (from) the remote user space directly. Suppose that the receiver process has allocated a buffer to store incoming data and the sender has allocated a send buffer. Prior to the data transfer, the receiver must have sent its buffer address to the sender. Once the sender owns the destination address, it initiates a direct-deposit data sending. This task does not interfere with the receiver process. On the receiver side, it focuses on computation tasks, testing if new messages have arrived, or blocking until an incoming message event arises.

The need for a one-sided communication protocol has been recognized for many years. Some of these issues were initially addressed by the POrtable Run-Time Systems (PORTS) consortium [25]. One of the missions of PORTS was to define a standard API for one-sided communications. During the development, several approaches were taken toward the one-sided API. The first one is the *thread-to-thread* communication paradigm which is supported by CHANT [17]. The second one is the *remote service request* (RSR) communication approach supported by libraries such as NEXUS [13] and DMCS [11]. The third approach is a *hybrid* communication that combines both prior paradigms and supported by the TULIP [7] project. All these paradigms are widely used. For example, NEXUS supports the grid computing software infrastructure GLOBUS. MOL [10] extends DMCS with an object migration mechanism and a global namespace for the system. DMCS/MOL is used both in the Parallel Runtime Environment for Multi-computer Applications (PREMA) [5] and in the Dynamic Load Balancing Library (DLBL) [4].

In 1997, MPI-2 [21] (the second MPI standard) has been released with specifications including some basic one-sided functionalities. Although, many studies have integrated one-sided communications to optimize MPI [12]. In 1999, a new communication library called Aggregate Remote Memory Copy Interface (ARMCI) [22] has been released. ARMCI is a high-level library designed for distributed array libraries and compiler run-time systems. IBM has maintained a low-level API, named LAPI [18], implementing the one-sided protocol and running on IBM SP systems only. Similarly, Cray SHMEM [6] provides direct send routines.

At the network layer, many factories have built RDMA features that make it easier the implementation of one-sided paradigms. For example, the HSL [29] network uses the PCI-Direct Deposit Component (PCI-DDC) [16] to offer a message-passing multiprocessor architecture based on a one-sided protocol. InfiniBand [19] proposes native one-sided communications. Myrinet [3, 9] and QNIX [28] do not provide native one-sided communications. But these features may be added (as for example in GM [1] for Myrinet since Myrinet NICs are programmable).

*GET / INT — CNRS UMR 5157 SAMOVAR, Département Informatique, 9, rue Charles Fourier, 91011 Évry, France, Tel: +33 1 60 76 45 73 — Fax: +33 1 60 76 47 80, {ouissem.benfredj, eric.renault} @int-edu.eu

The arrival of these kind of networks has imposed common message-passing libraries to support RDMA (GM [1], VIA [27], VAPI [20]...). Most of these libraries have been extended with one-sided communications to exploit RDMA features. But they do not use these functionalities as a base for their programming model.

2. RWAPI. Remote Write is a simple communication protocol that uses asynchronous communications and the one-sided paradigm as a programming model. The Remote-write protocol insure that the sender of a message provides all the information needed to copy a contiguous memory area from one node to another.

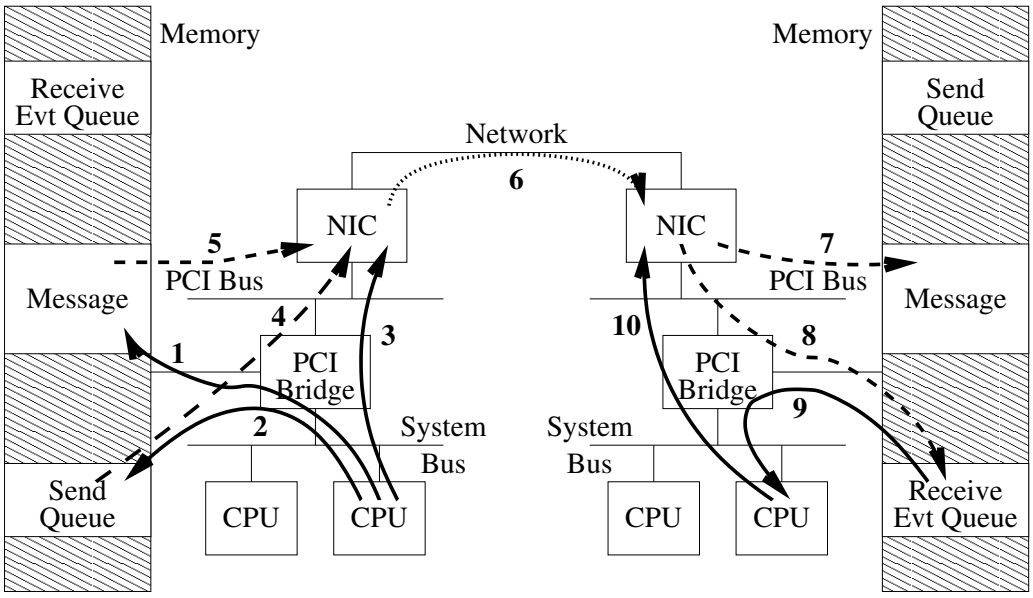


FIG. 2.1. *Message transfer with the remote-write protocol.*

Fig. 2.1 shows the different steps involved in a message transfer using the remote write:

1. the application writes the content of the message in a previously allocated contiguous memory area.
2. the application adds a new entry in the send queue. All information required to perform the message transfer are provided: the local address (where the content of the message is located), the remote address (where the message will be copied on the receiver) and the size of the message.
3. the application informs the NIC that a new entry has been posted in the send queue.
4. the NIC reads the next available entry in the send queue.
5. using the local address and the size of the message, the NIC reads the message content from memory...
6. and sends it to the destination node using the network.
7. when the message arrives on the destination node, it is copied in memory using the remote address and the size of the message.
8. once the message has been received completely, the NIC adds a new entry in the receive event queue.
9. the application is then able to read the content of the receive event queue to take into account new income messages.
10. when a message has been taken into account, the application notifies the NIC to free resources in the receive event queue.

RWAPI (which stands for Remote-Write Application Programming Interface) is a lightweight interface designed to provide a tiny API for the single remote-write primitive. The goal we are trying to achieve is to provide the smallest set of functions that enables to write any parallel programs. This way, we expect:

- to achieve the best performance for communications.
- to require as less development as possible to port our interface to new architectures (and GRWA — see next section — has been defined in this way).

There are two kinds of messages in RWAPI. The first message type requires the destination node identification, both local and remote addresses and the size of the message. Messages in this case can be of any length. The second message type just requires the destination node identification and the message content;

these messages do not require any addresses or size, but are limited to a few 16 bytes. They can be helpfully used to transfer small amounts of information of any kind from one node to another. However, even if they are not limited to this specific use, they are especially useful to exchange addresses before the other message type transfers can occur.

The API is as follows. Unless otherwise mentioned, the value returned by the functions is an error code:

- int rwapi_init (int, char **) must be called before any other RWAPI functions in order to set up the communication interface.
- int rwapi_finalize ( ) should be called after any other RWAPI functions and before exiting the program. This function ensures that all FIFOs are flushed before leaving.
- int rwapi_size ( ) returns the number of nodes in the virtual machine.
- int rwapi_rank ( ) returns the rank of the local node in the virtual parallel machine in the range from 0 to size minus 1.
- void * rwapi_alloc (size, net *) allocates a contiguous memory block of the given size in the virtual address space of the process. If the underlying network interface requires the use of contiguous physical memory, it is attached to the application transparently. The value returned by this function is the virtual address in the virtual address space of the process where the contiguous memory block has been attached. The second parameter is the address where the "network" address will be stored when returning from the function. This address is the one that must be used for transferring data.
- int rwapi_free (void *) deallocates the memory area provided as a parameter.
- int rwapi_send (node, small) sends a small message to another node.
- int rwapi_receive (node *, small *) returns information about the oldest incoming message that has not been received whatever the sender. The value returned by the function is 0 if there is no message pending and 1 if a message has been returned. In this case, both the node and the content of the small message are stored at the addresses provided as parameters.
- sid rwapi_write (node, net, net, size) sends an arbitrary-long message to another node. Both local and remote "network" addresses must be provided together with the size of the message. The value returned by this function is an SID (Send ID).
- int rwapi_issent (sid) returns whether the message identified by the SID has been sent or not. This is useful to determine when a memory area can be reused.

Note that, rwapi_ssend, rwapi_send, rwapi_issent and rwapi_receive are all non blocking functions.

3. GRWA. Even if lightweight, RWAPI is generic enough to be implement on top of any computer and communication architectures. In order to avoid rewriting several times the same pieces of code, we developed GRWA (the Global Remote-Write Architecture). GRWA is composed of a set of eight inter-dependable modules. Each module provides a specific set of services which can be used by the others. GRWA allows an application to run on a grid with heterogeneous nodes interconnected with different network hardware. Thus, it guarantees good levels of scalability and portability. Figure 3.1 presents the architecture of GRWA and the relations between the different modules (arrows on the figure represents the dependencies between modules).

Module ARCH. aims at providing a generic interface for processor specific functionalities. Typically, this includes the ability to read/write on I/O ports: almost all processors provides a way to let users access I/O ports; however, processor access and operations may vary from one processor to another.

Module BUS. implements a separate module for bus management. This allows portability as some NICs are available for more than one I/O bus, and more than one NIC is available for a given I/O bus. It can also improve the efficiency of the interfaces provided by GRWA as it allows to replace the operating-system implementation by any customized ones.

Module CRYPT. aims at ensuring the security of "network" addresses used to perform message transfers. At the physical layer, NICs are using physical addresses instead of virtual ones. A solution would consist in translating virtual addresses to physical addresses each time a transfer occurs. However, as whatever the way it is performed, the translation is costly, we consider it is better to provide users the addresses involved in message transfers. Then, provisions have to be made in order to check the validity of "network" addresses provided by the user.

Module INTERFACE. provides users a unified API for message passing. This is the only module programmers should see. RWAPI is the default API provided by GRWA. However, other interfaces may be provided. For example, [14] shows it is possible to provide an efficient implementation of other message-passing libraries (like MPI) using only the remote-write primitive.

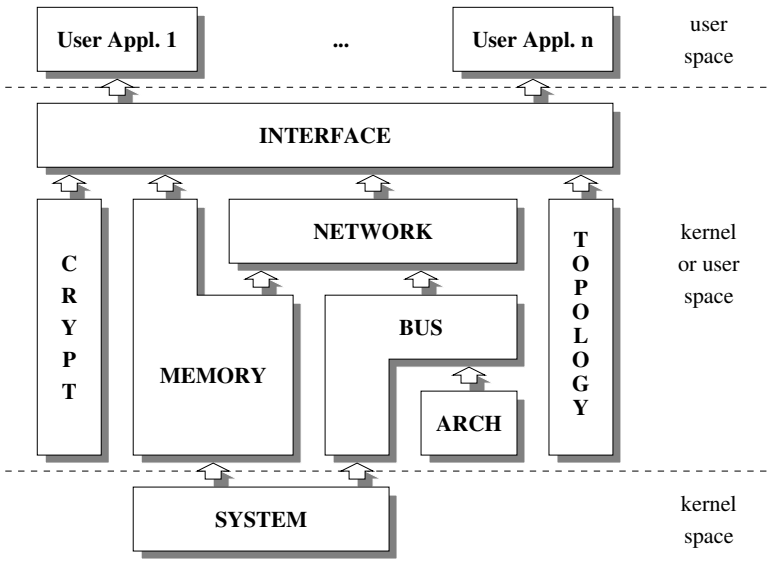


FIG. 3.1. *Global Remote-Write Architecture.*

Module MEMORY. manages the memory used to transfer information between nodes. As memory areas used to send and receive data are not necessarily a multiple of a page size, and as memory is a critical resource (especially when the network interface requires the use of physical contiguous memory blocks), it is important to manage the memory resource carefully. As a result, this module manages physical and virtual memory undistinctively, in a similar way as the malloc-free couple of functions does.

Module NETWORK. is in charge of managing NICs and routers. In order to improve performance, this management should be performed at the lowest level. This is the case for the implementation [23] on top of the Multi-PC machine [15] (using a HSL network [29]) which requires the application to deal directly with the on board NIC component. However, it is possible to provide implementations on top of any other message-passing libraries, as long as the NETWORK module API is respected. This is useful to provide our interfaces on top of any new architecture until a native implementation becomes available.

Module SYSTEM. aims at exporting data from kernel space to user space or at providing applications an access to functionalities which are traditionally reserved to priviledged users.

Module TOPOLOGY. provides a set of functions to determine which node number is attached to a hostname, the number of nodes in the parallel virtual machine and useful topological information to route messages.

4. InfiniBand. According to the last list of the TOP500 [2, 24], InfiniBand is the third most used interconnect. The InfiniBand Architecture (IBA) [19] is a new industry-standard architecture for server I/O and inter-server communication. It was developed by the InfiniBand SM Trade Association (IBTA) to provide the levels of reliability, availability, performance, and scalability necessary for present and future server systems, levels significantly better than what can be achieved with bus-oriented I/O structures.

The paradigm of the InfiniBand Architecture is message-passing. It incorporates many of the concepts of the Virtual Interface Architecture. Each vendor had a different software stack with a proprietary added value. However, there are multiple vendor-independent access layers that support different HCA (Host Channel Adapter or network card) simultaneously. A vendor-independent access layer is called a verb.

Figure 4.1 shows that the software stack of the Mellanox IB-Verbs API (VAPI [20]) interface provides a set of operations that closely parallel the proto verbs of the InfiniBand standard, plus additional extension functionalities in the areas of enhanced memory management and adapter properties specifications.

In addition to the above interfaces that can be used to communicate with the Host Channel Adapters' provider drivers directly, there also exist portable interfaces that hide the channel access interface from the user point of view. These include SRP, IPoIB, SDP, and SM. SRP (the SCSI RDMA Protocol) enables access to remote storage devices across an InfiniBand fabric. IPoIB provides standardized IP encapsulation over InfiniBand fabrics as defined by the IETF. This driver operates only within an InfiniBand fabric. SDP (the Sockets Direct Protocol) is an InfiniBand specific protocol defined by the Software Working Group (SWG) of

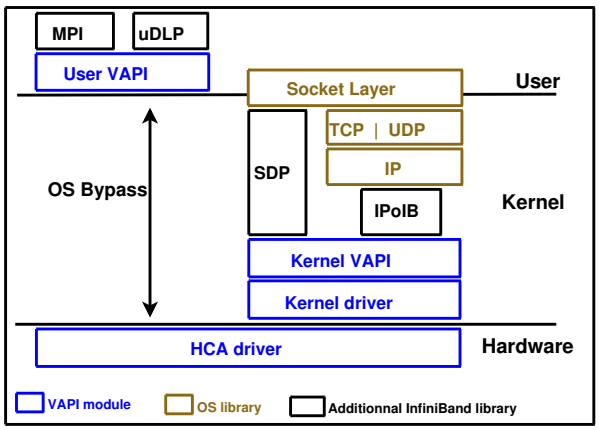


FIG. 4.1. *The Mellanox InfiniBand-Verbs API (VAPI) software stack.*

the InfiniBand Trade Association (IBTA). It defines a standard wire protocol over IBA fabric to support user-mode stream sockets (SOCK_STREAM) over IBA. SM (the Subnet Manager) handles several areas related to the operation of the subnet including: discovery, monitoring and configuration of the ports connected to the subnet, responding to subnet administrative (SA) queries, configuration of I/O units with host channel drivers, performance management, and baseboard management.

InfiniBand is an I/O channel based architecture [26] and not a register based one. A channel based architecture offloads the interprocessor communication overhead from the CPU to provide the maximum available performance. For large multiprocessor applications the performance gain is significant. The use of DMA engines at every InfiniBand node is critical to offloading the CPU. In a traditional load/store model, the data must pass through all levels of the memory hierarchy on its way to and from a CPU register. The device driver is responsible for maintaining the queue of transfers for the device. In an InfiniBand based system, I/O operations are scheduled as queued DMA operations and are handled by the node hardware rather than the CPU. In addition to preserving memory bandwidth and cache resources, this architectural feature also minimizes the number of interrupts that must be serviced by the CPU during a data transfer.

The primary architectural element is the Queue Pair (QP). Each QP consists in an outbound queue, and an inbound queue. Queue Pairs are not shared between applications; therefore, once they are set up, they can be managed at the application level without incurring the overhead of system calls. The QP is the mechanism by which quality of service, system protection, error detection and response, and allowable services are defined. An application can use many QPs, each one with a different quality of service. Creating a QP requires support from the operating system which handles the HCA and initialize memory regions to be used by QPs to manage communication requests and memory operations.

The following transfer types are possible between Queue Pairs: *Send* which includes a scatter/gather capability; *RDMA-Write*; *RDMA-Read*; *Atomic Operations* like Compare & Swap and Fetch & Add in remote memory; *Bind Window* for remote memory management; and *Multicast*.

Each QP is configured for a particular type of service independently. These service types provide different levels of service and different error recovery characteristics. The available transport service types include: Reliable Connection (RC), Unreliable Connection (UC), Reliable Datagram (RD), Unreliable Datagram (UD) and Raw.

5. InfiniWrite: the RWAPI interface over InfiniBand. This section presents the design of InfiniWrite which is an implementation of RWAPI over InfiniBand. The InfiniBand architecture is a complex and hierarchical structure comprising many components, libraries, programming interfaces, and modules along with multiple protocols that can be used based on application requirements. Thus, a native implementation of RWAPI over InfiniBand was not planned. Although, we chose to implement RWAPI on top of VAPI which is the lowest reliable layer of the Mellanox software stack.

Like any software and middleware developed on top of InfiniBand, RWAPI processes must use the same InfiniBand type of service. At this time, InfiniWrite uses RC which guarantees the highest level of reliability. However, the RC service requires a complex initialization and a big amount of NIC memory. Actually,

rwapi_initialize() creates $(N-1)$ QPs (N is the number of processes for the RWAPI application). Then, it sends the QP numbers to remote processes, one QP number for each remote process. In order to do so, it calls the *exchange* collective operation provided by the bootstrap module called *Fughara*. The initialization function also communicates the mapping process identifier (LID) to the other processes.

In this context, any external memory module could not be used since the VAPI memory module is devoted to the NIC driver. In fact, the VAPI_register_mr() function pins the memory pages as requested for subsequent DMA transfers, translates virtual addresses of the pinned pages into physical addresses, and transmits the obtained physical addresses to the NIC on board driver.

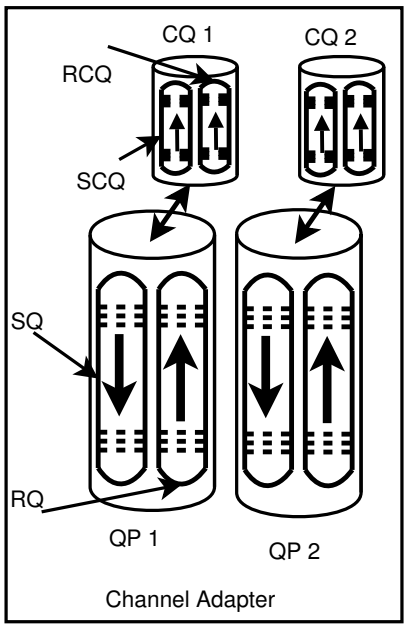


FIG. 5.1. *The organization of the InfiniWrite communication queues.*

Communications with VAPI are regulated by the previously allocated QPs. Each QP encompass a send work queue (SWQ) and a receive work queue (RWQ). The RC type of service supports all transfer types (SEND, RDMA-Read, RDMA-Write, Atomic Operations, and Bind Window). Sending a small message with rwapi_send() can be performed easily by EVAPI_post_inline_sr() which posts a small amount of data in the SWQ using the *Send* transfer type and the PIO mechanism (without any virtual to physical memory address translation). Rwapi_write() takes advantage of the *RDMA-Write* transfer type. In the case of VAPI, the "network" address which describes a rwapi-allocated memory region is mapped into a structure containing the virtual address and two other VAPI-related information, ie. a key and a memory region number.

Since the SWQ size can be set up to a great number of entries (several thousands), there is no need for a host send queue. The InfiniWrite's send functions are non-blocking operations as long as the SWQ is not full. When the SWQ overflows, which is quite infrequent, the function blocks until a send request is executed and a SWQ entry is freed.

InfiniBand is based on the rendez-vous communication model which requires that a receive request is posted before the corresponding send request. Thus, InfiniWrite associates a pre-allocated receive buffer to each entry of the $(N-1)$ RWQs. When a small message arrives, InfiniWrite copies the received data to the user destination, free the receive buffer, and posts it again in the same RWQ entry. In order to perform this last task, InfiniWrite tags each receive buffer with its associated RWQ entry. Then, when a new receive arises, InfiniWrite uses the receive buffer which tag is the RWQ entry number.

In order to control message reception, InfiniWrite sets up at least $(N-1)$ receive completion queue (RCQ), one RQC for each remote process. Rwapi_receive() then polls on the $(N-1)$ RCQ to check if a message has been received. Note that VAPI provides a interrupt-driven receive control. However, InfiniWrite does not use this feature because the management of interrupts is very expensive. This is especially costly for small messages because the interrupt time is comparable with the total communication time.

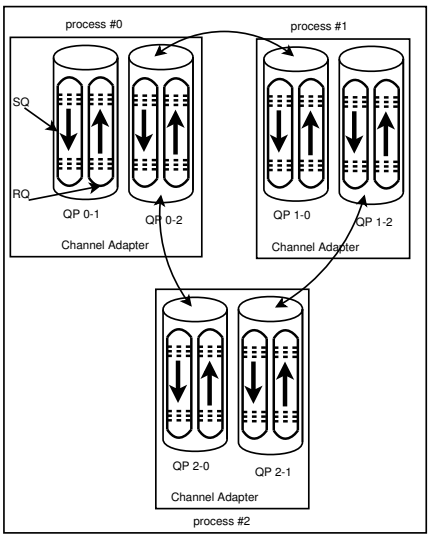
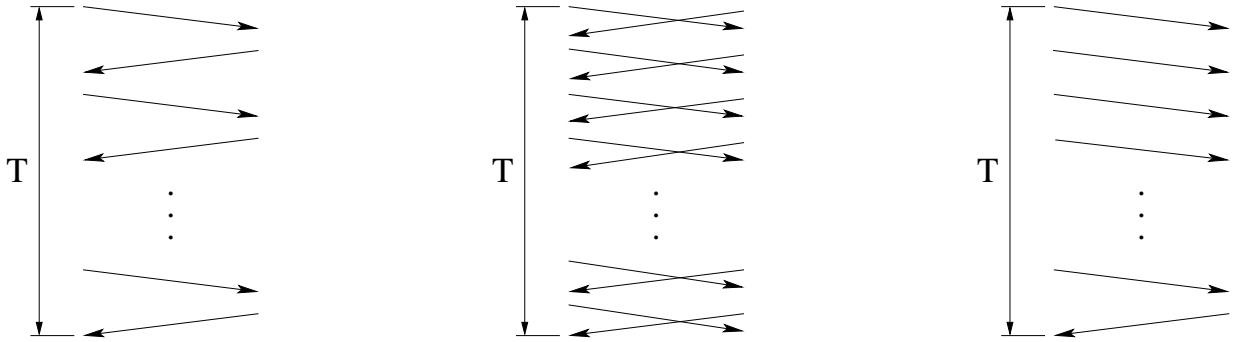


FIG. 5.2. *InfiniWrite design using three processes.*

InfiniWrite implements Rwapi_issent() with $(N-1)$ send completion queues (SCQ). Each time it is called, rwapi_issent() polls SCQs and increments a global variable which stores the total number of the send requests performed locally. The function returns *true* if the value of the argument is greater than the value of the global variable.

6. Performance measurements. We compared our implementation with two other existing libraries on top of the InfiniBand Technology: VAPI, the native interface available on top of InfiniBand, and MPI, developed on top of VAPI for InfiniBand. In order to make the comparison between these communications libraries, we use three separate benchmarks (see Fig. 6.1).



(a) Round-trip. (b) Bidirectional RT. (c) One-way.

FIG. 6.1. *Performance benchmarks.*

The first benchmark (Fig. 6.1(a)) is the classic ping-pong in which a message can be sent only once the previous one has been received. This benchmark is used to determine the one-way latency of the network for various message size. The second one (Fig. 6.1(b)) is a bidirectional ping-pong which is used to highlight if a library takes benefits of the bidirectional links. The last benchmark (Fig. 6.1(c)) is the burst. Data is flowing in one direction only and the sender does not need to wait acknowledgment from the receiver before it sends the next one. This benchmark essentially measures the reciprocal of the gap g (the interval between successive sends when the buffering capacity of the network is saturated).

Performance have been measured on Muse, one of the clusters of the Institut National des Télécommunications. Muse is composed of 16 nodes connected using both an InfiniBand 4× interconnection network for data and a Gigabit Ethernet interconnection network as a control network. Each node includes a 1.8-GHz AMD

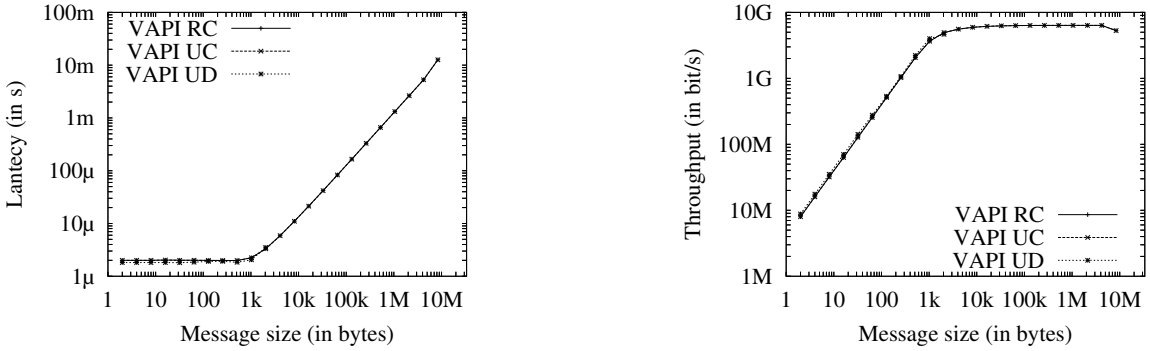
Opteron processor with a 1-MB cache and 2 GB of memory. For mass storage, each node is associated a 40-GB IDE hard drive, except for the first node (the front-end node) which is associated a 80-GB IDE hard drive; note that users accounts are stored on the front-end.

The measurement protocol is as follows: for each message size, each benchmark is run ten times. The duration of a run is one minute (this ensures a high consistency in results and we have determined that the confidence interval is greater than or equal to 90%). The system time is registered before the first message is sent (t_1) and after the last message is received on the same node (t_2). Let the elapsed time t be the difference between both. For a given run, let s be the size of messages and n be the number of effectively transmitted messages.

Let the end-to-end latency L (in the following we use the term latency) be the ratio between the elapsed time t and the number of effectively transmitted messages n. And let the user throughput T (in the following we use the term throughput) be the ratio between the amount of data (number of effectively transmitted messages n times the size of a message s) and the elapsed time t.

$$t = t_2 - t_1 \qquad L = \frac{t}{n} \qquad T = \frac{n \times s}{t}$$

Fig. 6.2 provides both latency and throughput for all three service types. These graphs show that they all roughly provide the same communication performance. As a result, we chose to use RC only to make the comparison with the other two libraries (MPI and our remote-write message-passing interface RWAPI) as it is easy to work with and provides reliable communication.



(a) Latency. (b) Throughput.

FIG. 6.2. *VAPI performance.*

Fig. 6.3 presents a comparison of performance for VAPI, MPI and InfiniWrite message-passing libraries. Fig. 6.3(c) and Fig. 6.3(f) show that InfiniWrite performance are very close to the native VAPI performance.

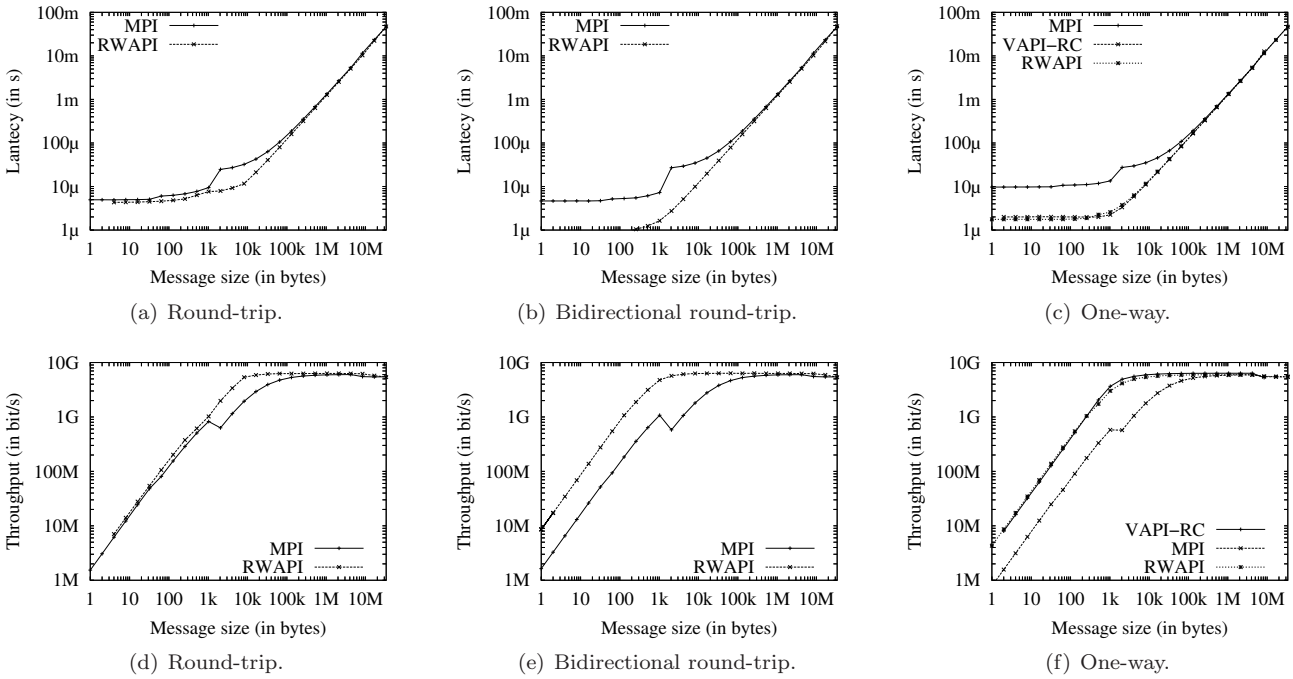
In a general way, Fig. 6.3 shows that InfiniWrite performance are always better than MPI performance. More specifically, the maximum ratio between the minimum latency achieved by RWAPI and the minimum latency achieved by MPI is up to 5.5 μs for small messages (i. e. 1.76 μs for InfiniWrite and 9.71 μs for MPI using the one-way benchmark).

Both InfiniWrite and MPI are able to achieve the maximum user throughput for long messages. However, InfiniWrite is able to provide this maximum user throughput for messages as short as 4 kilo-bytes while MPI cannot do the same for messages smaller than few hundreds kilo-bytes.

InfiniWrite, as a lightweight interface, coupled with the InfiniBand interconnect, which is characterized by a low-latency routing, provides a good area for scalability in terms of number of nodes.

Finally, the curves on Fig. 6.3 shows that there is an important difference in the management of short and long messages for MPI, represented with a significant decrease of the throughput between 1 and 2 kilo-bytes.

7. Related Works. Besides RWAPI, there are other high-performance libraries which are implemented over InfiniBand. Some of the examples use one-sided communications as a programming model and the others use two-sided communications. The first class includes ARMCI GASNET and the extended version of MPI-2. ARMCI is a communication library that optimizes communications in the context of distributed arrays. It



(a) Round-trip. (b) Bidirectional round-trip. (c) One-way.

(d) Round-trip. (e) Bidirectional round-trip. (f) One-way.

FIG. 6.3. *Performance comparison.*

is suited only for fine-grained communications like with a distributed compiler. GASNET provides a global shared-memory abstraction to the user, regardless the hardware implementation and based on the Active-Messages paradigm with a modified Interrupt implementation. However, Active-Messages threads add an extra overhead due to the creation of threads and their management in addition to the thread-safe problem. MPI-2 extends MPI with some one-sided communication routines. The second class of libraries mainly includes the various implementations of MPI. These implementations do not benefit from RDMA capabilities. Recent studies [14] have tried to implement the MPI interface with one-sided routines but the performance is still lower.

8. Conclusion and Future Works. In this paper, we have proposed a design for RWAPI over InfiniBand called InfiniWrite. This design takes full advantages of the InfiniBand hardware such as OS-Bypass and RDMA, thus eliminating the involvement of the operating system and the receive process. In addition, InfiniWrite allows the overlap between communication and computation.

In order to decrease the end-to-end latency of small messages, we used the Programmed-IO facility instead of RDMA to copy data to the network card, so as to remove one long-startup-time DMA transaction.

Through performance evaluation, we have shown that our design can achieve a low latency (about 1.76 μs) and a high user throughput (more than 6300 Mb/s, ie. the maximum available bandwidth for the user) even for short messages. As a comparison, the lowest latency provided by MPI over the same platform is 4.96 μs and the maximum user throughput cannot be achieved for messages smaller than several hundreds of kilo-bytes.

The lowest InfiniBand communication layer (VAPI) let the user choose between producing events for transfer operation completion or not. This does not suit RWAPI as disabling events does not allow the user to be notified for the completion of send operations and enabling events adds an extra overhead due to the unnecessary receive completion.

Currently, InfiniWrite uses RC as the type of service provided by the InfiniBand packet management. RC requires a connection between each remote HCA and thus consumes much HCA memory resources. Consumed memory is mainly used to store data reassembly informations for each connection. To achieve better scalability, we are working on applying the RD type of service which bypasses any connection management and maintains a reliable communication.

REFERENCES

[1] *GM: A message-passing system for myrinet networks 2.0.12*, 1995.
[2] *TOP500 list.* `http://www.top500.org` November 2006.
[3] ANSI/VITA 26-1998, *Myrinet-on-VME Protocol Specification.*, 1998.
[4] M. Balasubramaniam, K. Barker, I. Banicescu, N. Chrisochoides, J. Pabico, and R. Cariño, *A novel dynamic load balancing library for cluster computing*, in Proceedings of the IEEE International Symposium on Parallel and Distributed Computing (ISPDC/HeteroPar 2004), Cork, Ireland, July 2004, IEEE Computer Society, IEEE, pp. 346–353.
[5] K. J. Barker, *Runtime support for load balancing of parallel adaptive and irregular applications*, PhD thesis, 2004. Adviser-Nikos Chrisochoides.
[6] R. Barriuso and A. Knies, *SHMEM User's Guide*, Cray Research Inc, 1994.
[7] P. Beckman and D. Gannon, *Tulip: Parallel run-time support system for pC++.*
[8] J. Beecroft, D. Addison, F. Petrini, and M. McLaren, *Quadrics QsNet II: A network for Supercomputing Applications*, in In Hot Chips 15, Stanford University, Palo Alto, CA, August 2003.
[9] N. Boden, D. Cohen, R. Flederman, A. Kulawik, C. Seitz, J. Selzovic, and W. Su, *Myrinet—A Gigabit-per-Second Local-Area Network*, vol. 15, 1995, pp. 29–36.
[10] N. Chrisochoides, K. Barker, D. Nave, and C. Hawblitzel, *Mobile object layer: A runtime substrate for parallel adaptive and irregular computations*, Advances in Engineering Software, 31 (2000), pp. 621–637.
[11] N. Chrisochoides, I. Kodukula, and K. Pingali, *Data Movement and Control Substrate for parallel scientific computing*, in First International Workshop on Communication and Architectural Support for Network-Based Parallel Computing, D. Panda and C. Stunkel, eds., vol. 1199 of Lecture Notes in Computer Science, San Antonio, TX, February 1997, IEEE Computer Society, Springer-Verlag, pp. 256–268.
[12] J. Dobbelaere and N. Chrisochoides, *One-sided communication over MPI-1.*
[13] I. Foster, C. Kesselman, and S. Tuecke, *The Nexus task-parallel runtime system*, in Proc. 1st Intl Workshop on Parallel Processing, Tata McGraw Hill, 1994, pp. 457–462.
[14] O. Glück, *Optimisations de la bibliothèque de communication MPI pour machines parallèles de type "grappe de PCs" sur une primitive d'écriture distante*, PhD thesis, Université Paris VI, July 2002.
[15] O. Glück, A. Zerrouki, J. Desbarbieux, A. Fenyo, A. Greiner, F. Wajsbürt, C. Spasevski, F. Silva, and E. Dreyfus, *Protocol and Performance Analysis of the MPC Parallel Computer*, in Proceedings of the 15th International Parallel and Distributed Processing Symposium, San Francisco, CA, April 2001, pp. 52–58.
[16] A. Greiner, J. Desbarbieux, J. Lecler, F. Potter, F. Wajsbürt, S. Penain, and C. Spasevski, *PCI-DDC Specifications*, Laboratoire MASI, Université Paris VI, September 1996.
[17] M. Haines, P. Mehrotra, and D. Cronk, *Chant: Lightweight threads in a distributed memory environment*, 1995.
[18] IBM Corporation, ed., *LAPI Programming Guide*, no. IBM Document Number: SA22-7936-00 in IBM Reliable Scalable Cluster Technology for AIX L5, First Edition, Poughkeepsie, NY, September 2004.
[19] InfiniBand Trade Association, *The InfiniBand Architecture, Specification Volume 1 & 2*, June 2001. Release 1.0.a.
[20] Mellanox Technologies Inc, *Mellanox ib-verbs api (vapi)*, 2001.
[21] Message Passing Interface Forum MPIF, *MPI-2: Extensions to the Message-Passing Interface.* Technical Report, University of Tennessee, Knoxville, 1996.
[22] J. Nieplocha and B. Carpenter, *ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems*, Lecture Notes in Computer Science, 1586 (1999), pp. 533–546.
[23] E. Renault and P. David, *Performance and Analysis of the PCI-DDC Remote-Write Implementation*, in 2001 IEEE International Conference on Cluster Computing, D. Katz, T. Sterling, M. Baker, L. Bergman, M. Paprzycki, and R. Buyya, eds., Newport Beach, CA, October 2001, pp. 197–203.
[24] E. Strohmaier, *Top500—top500 supercomputer*, in SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing, New York, NY, USA, 2006, ACM Press, p. 18.
[25] The PORTS Consortium, *The PORTS0 Interface*, Technical Report ANL/MCS-TM-203, Mathematics and Computer Science Division, Argonne National Laboratory, February 1995.
[26] Tom Shanley, *InfiniBand Network Architecture*, PC System Architecture, MindShare, Inc., 2003.
[27] *Virtual Interface Architecture Specification, Version 1.0, published by Compaq, Intel, and Microsoft.* December 1997.
[28] Amelia De Vivo, *A Light-Weight Communication System for a High Performance System Area Network,* Universitá di Salerno—Italy, November 2001.
[29] C. Whitby Strevens and al., *IEEE Draft Std P1355—Standard for Heterogeneous Interconnect—Low Cost Low Latency Scalable Serial Interconnect for Parallel System Construction*, 1993.

Edited by: Dana Petcu
Received: May 31, 2007
Accepted: June 17, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 281–290. http://www.scpe.org
ISSN 1895-1767

AN AGENT-BASED APPROACH TO GRID SERVICE MONITORING

KEITH ROCHFORD, BRIAN COGHLAN AND JOHN WALSH*

Abstract. The centralised management of distributed computing infrastructures presents a number of considerable challenges, not least of which is the effective monitoring of physical resources and middleware components to provide an accurate operational picture for use by administrative or management staff. The detection and presentation of real-time information pertaining to the performance and availability of computing resources is a difficult yet critical activity.

This architecture is intended to enhance the service monitoring experience of a Grid operations team. We have designed and implemented an extensible agent-based architecture capable of detecting and aggregating status information using low-level sensors, functionality tests and existing information systems. To date it has been successfully deployed across eighteen Grid-Ireland sites.

1. Introduction. Effective monitoring of complex distributed infrastructures, such as electricity supply and traffic management systems, is essential if expected levels of service are to be achieved. Operations Centres are frequently established to satisfy this requirement, providing a central authoritative point of operational information. They are, however, only as good as the underlying information systems they depend on.

Distributed computing infrastructures such as Grids have similar monitoring requirements. For those responsible for the management and administration of the individual resources or the infrastructure as a whole it is desirable that some form of monitoring system is put in place to highlight problems and abnormalities. Centrally managed infrastructures, such as Grid-Ireland[17] or EGEE[3], present additional challenges. Large quantities of status information from disparate resources must be detected, aggregated and stored.

We describe the principles of and justifications for the monitoring system currently deployed on the Grid-Ireland infrastructure and provide a technical overview of the sensors, communication and presentation systems which make the aggregate real-time status information available to the members of the operations team. The monitoring information is presented in such a manner as to facilitate the management of the infrastructure in response to status and security events, and provide a tool to maximise the impact of public relations activities and demonstrations.

2. Grid Monitoring. Grid monitoring may be described as the activity of measuring significant grid related resource parameters in order to analyse usage, availability, behaviour and performance[12]. Its objectives include:

- Locating performance problems
- Tuning for better performance
- Fault detection and recovery
- Input to prediction services

The size and nature of grid systems means that monitoring systems must be capable of measuring a wide range of metrics relating to a substantial number of entities distributed across a large geographical area. Monitoring activities present those responsible for the maintenance of the infrastructures with considerable challenges and has become an active area of research in it's own right[22]. Traditional network and host monitoring tools often lack the scalability, dynamicity, or extensibility required for effective deployment within computational or data grids.

3. Context. In order to better understand the motivation for this work we will briefly discuss the architecture of of the underlying infrastructure. Grid-Ireland is a managed layer providing grid services above the Irish research network. It allows researchers to share computing and storage resources using a common interface, and facilitates international collaboration by linking Irish sites to into European grid infrastructures being developed under such EU projects as EGEE[3], LCG[9], CrossGrid[2], and int.eu.grid[8].

The Grid-Ireland operations centre is based at Trinity College Dublin and is staffed by members of our research group. Among its responsibilities is the deployment and maintenance of an homogeneous core infrastructure[17] comprised of *grid gateways*. Installed at each site, these gateways provide a point of access both for external grid users to gain access to the site resources and local site users to gain access to remote grid resources.

*Department of Computer Science, Trinity College Dublin, Ireland. keith.rochford, {coghlan, john.walsh}@cs.tcd.ie

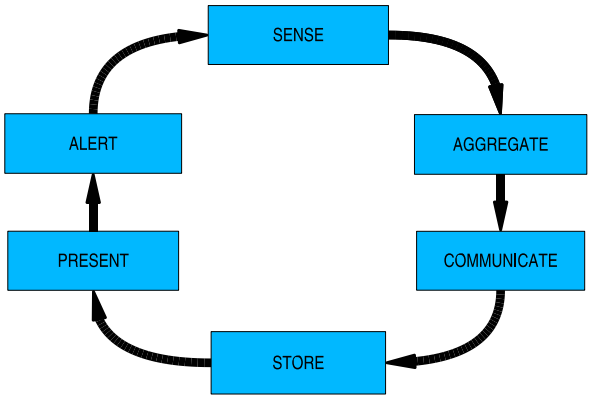


FIG. 2.1. *The Monitoring Cycle*

The gateway[15] provides the essential functionality of a grid site, thus freeing site administrators to concentrate on the management of their own resources. The gateways are remotely managed by the operations centre. Local cluster administrators need not concern themselves with the provision of grid services as these are an integral part of the deployed gateways. Resources at a site remain under the control of the site administrators who are free to manage as they see fit, without having to concern themselves with the finer details of grid integration.

Thus the core infrastructure of grid gateways remain the responsibility of the operations team. The ensuing possibility of homogeneity of this core infrastructure presents a number of opportunities to the users, grid operators and the site administrators. These include:

- Minimizing the proportion of software components that need to be ported to non-reference platforms.
- Maximizing the availability of the infrastructure while reducing the effort required for software deployment. The common software and hardware components facilitate centralised management and push-button transactional deployment of middleware components[18], guaranteeing uniform responses to management actions.
- Decoupling of grid infrastructure and site management. The fact that the site resources and the grid infrastructure are independent of each other allows for variation in design, deployment, and management.
- The installation of heterogeneous site resources based on non-reference architectures is also supported. In order to support a wide range of connected resources, the operations team carryout porting of the necessary middleware components[20].

Each grid gateway is composed of a set of seven machines: a firewall, an install server, a compute element (CE), a storage element (SE), a user interface (UI), a test worker node that is used for gateway testing, and optionally a network monitor. In general these are implemented as virtual machines on one physical host. All sites are identically configured with grid software based on gLite3. It is important to recognise that despite being deployed within remote networks, the management and availability of these hosts and services remains the responsibility of the centrally located Grid Operations Team.

4. Centralised vs. Agent-based Monitoring. Remote entities are typically monitored using one of two approaches; a centralised approach where one or more processes executing at a single location are responsible for the monitoring of all entities or a federated approach involving multiple distributed processes and sensors. In a centralised system, all monitoring processes are executed from a single location and attempt to determine the status of monitored entities either through the establishment of some form of connection with the entity or by performing some operation involving it. Distributed monitoring architectures employ monitoring processes on or closer to the monitored entity. These processes, often referred to as agents, perform the required tests and report the outcomes, through push or pull mechanisms, to the interested parties. The area of software agents is one of active research and many types of agents have been identified or proposed. These range from simple processes to complex intelligent autonomous mobile elements of software. It is argued that the popularity of

the term has led to its misuse and it might therefore be appropriate to present an explanation of what should be inferred by the term in the context of this work. We will use the term 'agent' to refer to an independent software component that performs operations on behalf of a user or other piece of software and exhibits some degree of autonomic behaviour.

Both methods of monitoring have advantages and the choice of which is best suited to a given situation is dependent on a number of factors including the size and nature of the systems, the required information and the degree of control that is required. It is often claimed that one approach is superior to the other, however the merits of each must be evaluated with respect to the individual systems and monitoring requirements. While it is true that the deployment of distributed monitoring processes can incur additional effort it is not necessarily true that the configuration need be any more complex. Unfortunately, centralised monitoring typically implies a more limited depth of data gathering than is possible with remote agents. In addition the potential for the management and control of remote entities through interaction with the distributed agents makes them an attractive choice for complex infrastructures. Also the scale of distribution can vary. The requirement for the installation of a software agent on each monitored host is dependent on the motivation of the monitoring activity and the amount of information pertaining to each host that is required. If the motivation for the adoption of an agent-based approach is merely to overcome the limitations imposed by network access policies, it may be sufficient to deploy a single agent to each network segment, domain, or site.

A combination of both approaches might be considered the best solution in terms of grid monitoring. Remote agents provide a level of access and control that would otherwise be unfeasible, yet there are a number of advantages offered by a centralised monitoring approach. It is of limited value to know the internal status information pertaining to a particular resource if we cannot determine whether or not the resource is accessible and usable through the normal operational mechanisms. Furthermore, in the interest of consistency and conceptual simplicity a central monitoring process might be implemented as a monitoring agent deployed at the Operations Centre.

5. Motivations. Early grid service monitoring systems, such as MapCentre[13] and Ganglia[4], offered valuable but limited functionality, required complex configuration, and were employed with limited degrees of success on our particular infrastructure[17]. Monitoring traffic is often denied access into the remote networks hosting the monitored grid resources, greatly reducing the usefulness of these tools. The unsuitability of these systems prompted the investigation and assessment of alternatives.

Several existing host and network monitoring tools, including some supporting distributed monitoring, were assessed but did not offer a suitable solution. Many were intended for deployment within a single administrative domain. While changes to network settings, such as firewall rules, can be quickly authorised and actioned from within such environments, this is not true from outside the domain. From the viewpoint of an operations team the majority of the monitored resources reside within remote administrative domains which are beyond their control. In cases where the resources are remotely managed the remote network still remains beyond control; this makes efficient centralised monitoring of the resources impossible. This is the case for Grid-Ireland, which has a notably integrated remotely managed dedicated grid infrastructure, and in consequence a decision was taken to undertake research into this rather specialist monitoring field.

The monitoring solution described below employs a combination of distributed and centralised monitoring sensors along with the aggregation of information from existing monitoring tools and is specifically designed to speed the response to status and security events.

The immediate aims were to develop and deploy an extensible monitoring framework which would overcome the problems traditionally associated with the monitoring of distributed resources spanning multiple administrative domains to satisfy the information requirements of the operations team. The framework would take advantage of the considerable existing body of work carried out in the area of grid monitoring by aggregating existing systems and tools where provided, and allowing for the development of new sensors where none exist. A key concern was the administrative overhead involved in the deployment and management of the monitoring solution. While there were initially only six sites to monitor, the introduction of an additional twelve sites highlighted the need for an efficient solution.

6. Architecture. The architecture obeys the general principle of ensuring scalability by devolving work to the monitored nodes where possible. This is achieved through the combined use of monitoring agents local to each site, plus a central monitoring process. Traditional systems like Nagios[10] are configured at the site for the site, but this becomes an excessive burden. Here a more efficient integrated approach is taken.

6.1. Monitoring Agents. A single Monitoring Agent, implemented in Java, is deployed to each of the sites with the responsibility of executing service checks for hosts and services at that site. In addition to the distributed agents a central agent running at the Operations Centre determines metrics, such as site connectivity, and aggregates information from additional sources such as Site Functionality Tests[11], Grid Intrusion Detection Systems[21] and existing grid information systems, so that it may be included in the analysis and presentation process.

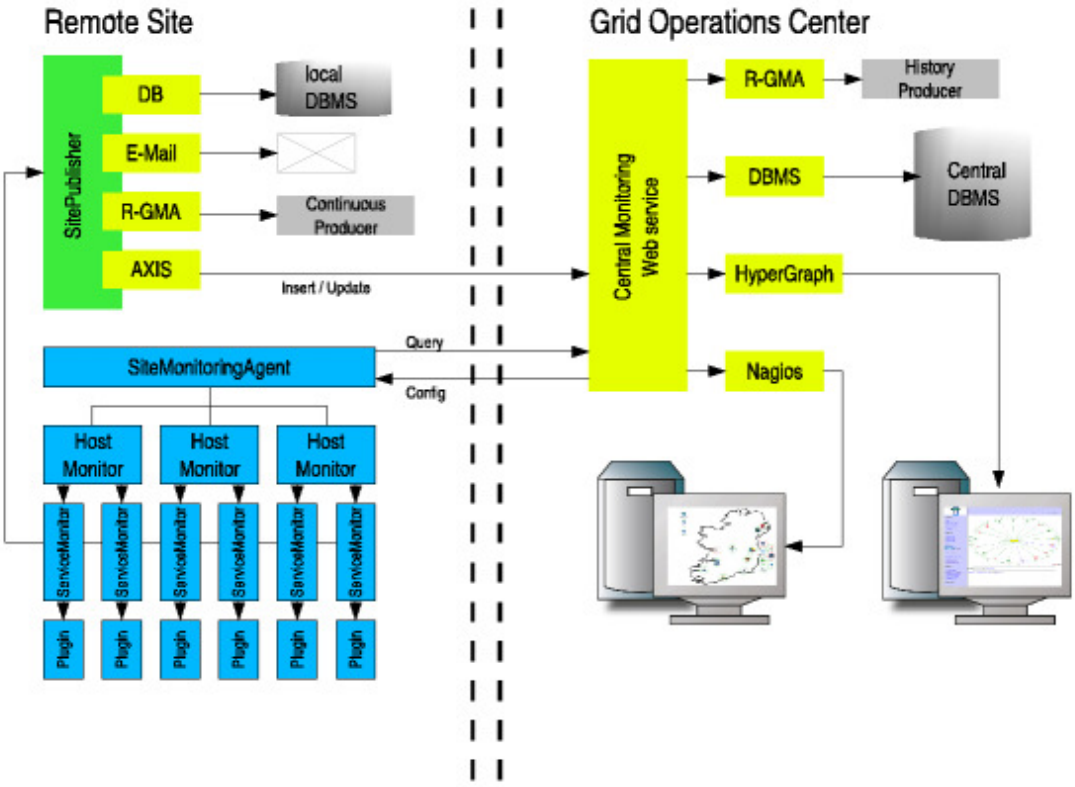


FIG. 6.1. *Agent-based monitoring architecture*

Configuration information specific to each site, host, or service is stored in a central database and made available to the agents via a web service, thereby minimising the amount of local site-specific configuration required. As described in Figure 6.2, upon start-up a remote agent queries the central configuration server for details of the hosts and services that it should monitor at its site. It then creates local monitoring objects that determine the availability of the specific service through the use of existing or custom service checking objects. At intervals specific to each service, the monitoring objects report their status to interested subscribers via a local publisher. One such subscriber is the central monitoring service, where the results are archived and, if necessary, brought to the attention of an operator. The use of remote sensors communicating over established ports (HTTPS) not only distributes the overhead associated with the monitoring operations but also eliminates many of the connectivity problems commonly experienced with centralised network monitoring.

In order to satisfy the requirement for extensibility, the status of each monitored entity is determined by the agent using plugin modules implemented as Java objects or as Nagios-compliant plugins written in either Perl, C, or as a shell script. This allows the agent to take advantage of a wide range of existing service checks developed for the Nagios project[10]. Examples of such custom plugins include log parsing mechanisms, information system consumers and mechanisms to query a Ganglia[4] monitoring daemon for system load information.

The collected status information includes but is by no means limited to:

- The tcp connectivity to grid service ports such as GRAM, GRIS etc.
- Network reachability of the machines comprising the grid gateway
- Queue / Job activity on the Compute Elements

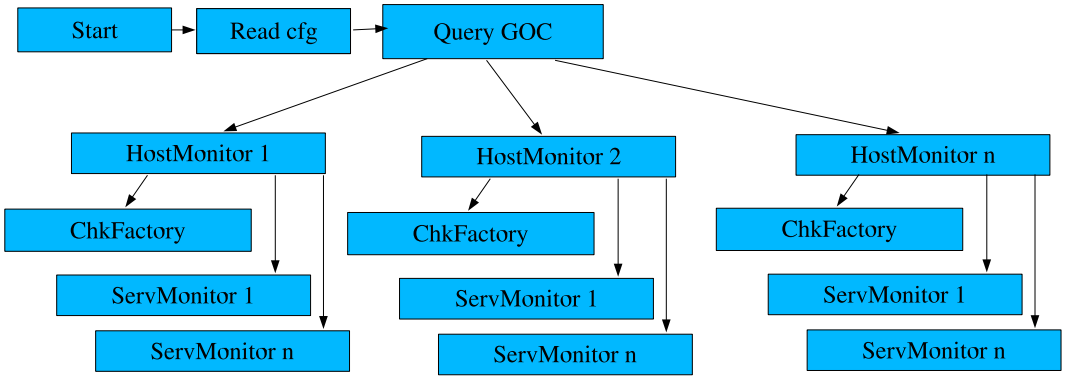


FIG. 6.2. *Monitoring Agent start-up sequence*

- The output of the Site Functionality Tests
- Grid Portal availability
- SSH reachability of the managed hosts
- Load information obtained via the Ganglia gmon daemon

6.2. Configuration Database. The configuration information defining what hosts and services should be monitored at each of the sites can be represented in text files or database tables, etc. In this case, advantage is taken of the database schema for version 1 of the LCG GOCDB[5] maintained at Rutherford Appleton Laboratory, UK, with several extensions. Hence the configuration information is stored in a mySQL database at the Operations Centre. A sub-set of the configuration database schema is illustrated in Figure 6.3. This shows the data stored for each host, service, and the data required to relate services and hosts for monitoring purposes.

6.3. Central Monitoring WebService. The central monitoring process, for entirely pragmatic reasons, needs to perform the following functions:

- provision of configuration information to each remote agent
- aggregation of information from remote agents and existing monitoring tools
- archiving and publishing of monitoring/status information

It is implemented as an AXIS[1] web service hosted at the Grid Operations Centre. Configuration information is made available to the monitoring agents via a web service employing the Jakarta Database Connection Pooling mechanisms to query the MySQL database. The use of this pooling mechanism to manage the database connections is an important part of ensuring the scalability of the central WebService. Since the Web Services are invoked frequently to transfer small amounts of information, establishing a database connection per session would result in slower response times and greater server load.

Using publication mechanisms similar to those embedded within the monitoring agents, status information reported back to the central server is made available to consumer APIs, via re-publisher mechanisms, or passed on to registered listeners. Currently implemented listeners include the Nagios NSCA client, JDBC, XML-RPC, and R-GMA[14] mechanisms. In addition to the central publisher mechanism, the status reports are cached in memory providing rapid access to the latest metrics for all monitored entities. Further optimizations are being explored.

7. Presentation and Alerting. Just as in many traditional Operations Centres, monitoring information is brought to the attention of the operations team in real-time by means of wall-mounted TFT displays, web-based tools and email/messaging alerting systems. Currently four 18 inch LCD display panels are used at the Operations Centre, but soon a number of 40 inch displays will replace these. In the near future, monitoring information and alerts will also be made available to the members of the operations team remotely through the use of mobile devices running web browsers and thin clients.

Nodes Table

Field	Type	Attributes	Null	Default
nodeID	mediumint(8)	UNSIGNED	No	
siteID	mediumint(8)	UNSIGNED	No	0
nodetype	varchar(50)		No	
nodetype2	varchar(50)		Yes	*NULL*
hostname	varchar(50)		No	
domain	varchar(200)		Yes	*NULL*
nagios_name	varchar(255)		Yes	*NULL*
ip	varchar(15)		Yes	*NULL*
grp	varchar(20)		Yes	LCG-1
hostdn	varchar(255)		Yes	*NULL*
system	varchar(50)		Yes	*NULL*
middleware	varchar(50)		Yes	*NULL*
install	varchar(50)		Yes	*NULL*
status	varchar(50)		Yes	*NULL*
operation	varchar(50)		Yes	*NULL*
comment	varchar(250)		Yes	*NULL*
monitor	enum('Y', 'N')		No	Y

Services Table

Field	Type	Attributes	Null	Default
serviceID	mediumint(9)		No	
serviceName	varchar(30)		No	
protocol	enum('TCP', 'UDP', 'PING', 'HTTP')		Yes	*NULL*
port	smallint(6)	UNSIGNED	No	0

Node-Services Mapping

Field	Type	Attributes	Null	Default
hostID	mediumint(9)		No	0
serviceID	mediumint(9)		No	2
checkPeriod	varchar(128)		No	24x7
maxAttempts	int(11)		No	3
checkInterval	int(11)		No	5
retryInterval	int(11)		No	3
contactGroups	varchar(255)		No	

FIG. 6.3. *A section of the Configuration Database Schema*

7.1. Reports/Displays/Alerts. The Nagios host and network monitoring system is employed at the operations centre for presentation and alerting, solely in order to take advantage of its web based display and reporting functionality in addition to its comprehensive alerting mechanisms. The configuration is purely passive in that Nagios is not responsible for any service monitoring directly but rather the information gathered from the remote sensors is fed into Nagios by means of the Nagios Service Check Acceptor server daemon, allowing the status of defined hosts and services to be updated.

7.2. Navigation of Information. For many existing host and network monitoring tools, navigation of the information is rather neglected. Not only is it desirable that this be easy, it should also be fast, so that an experienced operator can exhibit a "musicians touch", particularly in emergency situations. The most promising approach thus far assumes tree structures.

HyperGraph[6] is an open source project which provides Java code to work with hyperbolic geometry and in particular hyperbolic trees. Its extensive API facilitates the visualisation of graphs and hyperbolic trees which are extremely useful when dealing with large volumes of data in a hierarchical structure.

The HyperGraph API is used as an example of an alternative methodology for the display of status information, allowing the identification of problems 'at a glance' and manipulating the graph for an improved view of the necessary information. Navigation of the infrastructure tree is very fast, which is useful when attempting to find the location of an urgent problem.

In this case (see Figure 7.3), the colour of the edges between the Operations Centre node and the individual site nodes is determined by the network reachability of the gateway machine at that site. The status of the site nodes themselves is determined by the latest results of the Site Functionality Tests for that site. The colour of a host node is determined by the maximum alert value of the services monitored on that host. Nodes representing services are coloured based on the output of the pulgin for that service. The graph is constructed in such a way that hovering the pointer over a node causes further information pertaining to that node to be displayed. Table 7.1 defines the colour code used in the graph.

Although very useful, HyperGraph is not the panacea. It's assumption of tree structures is quite restrictive. Fast hierarchical navigation, particularly relational, is also very desirable and even necessary given our reliance on R-GMA. This is under investigation.

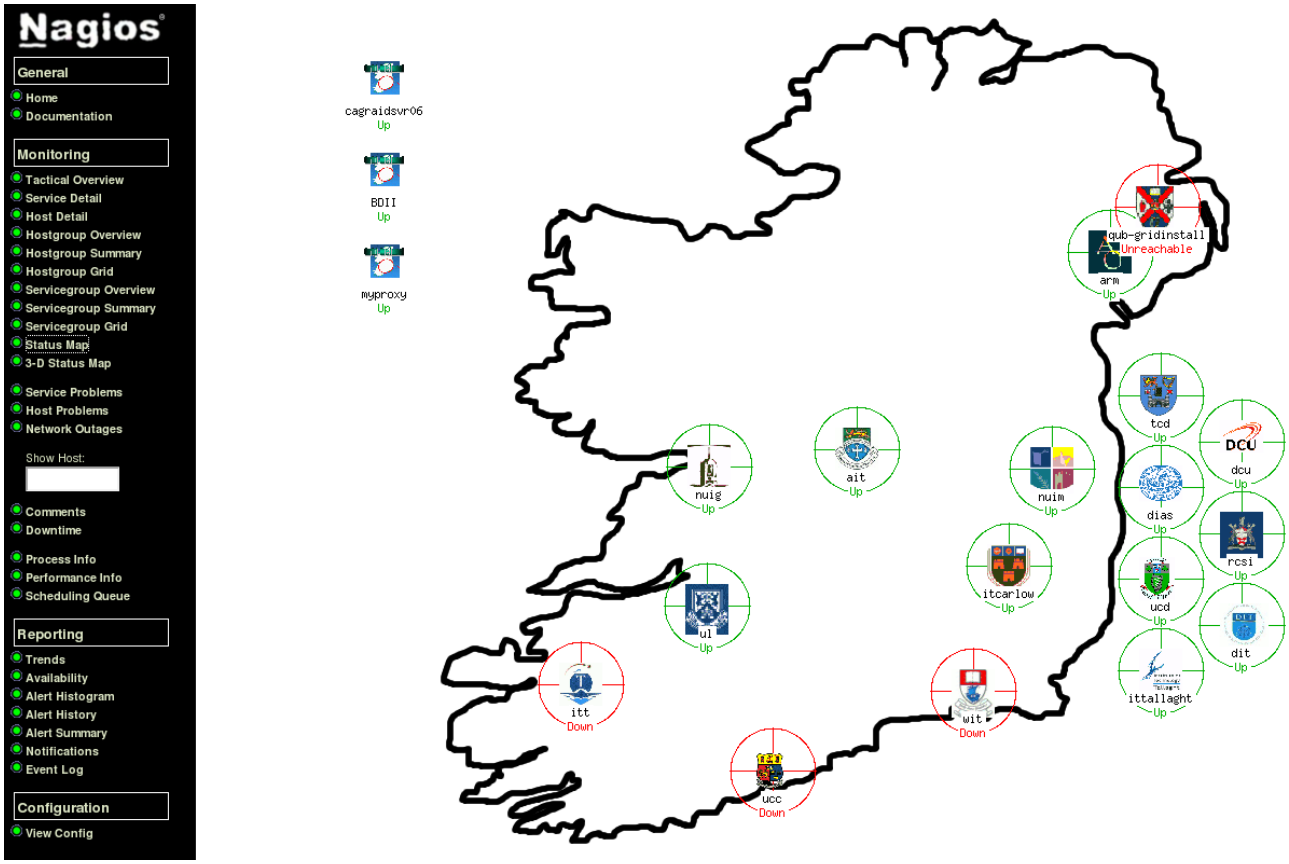
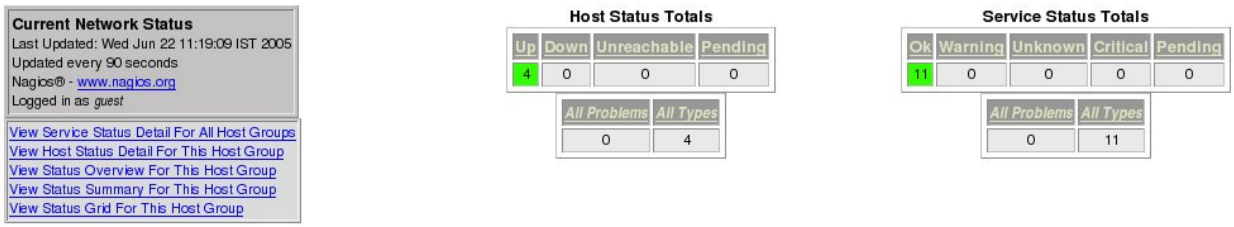


FIG. 7.1. *Example Nagios Display*



Service Status Details For Host Group 'tcd'

Host	Service	Status	Last Check	Duration	Attempt	Status Information
tcd-gridgate	GFTP	OK	06-08-2005 21:20:33	40d 18h 39m 28s	1/3	TCP OK - 0.459 second response time on port 2811
	GRAM	OK	06-08-2005 21:20:49	40d 18h 39m 28s	1/3	TCP OK - 0.387 second response time on port 2119
	GRIS	OK	06-08-2005 21:20:58	40d 18h 39m 28s	1/3	TCP OK - 0.345 second response time on port 2135
	SSH	OK	06-22-2005 11:15:38	16d 7h 7m 10s	1/3	SSH OK - OpenSSH_3.6.1p2-CERN20030917 (protocol 1.99)
tcd-gridinstall	APC	OK	06-22-2005 11:16:27	0d 7h 38m 9s	1/3	IN 228.9v (208.0 - 253.0) OUT 228.9v 50.0 Hz, Load 38.3 Percent, Charge 100.0 Percent, Batt 27.4v, Time left 33.0 mins, Internal Temp 31.5 C
	HTTP	OK	06-22-2005 11:17:09	49d 20h 40m 43s	1/3	HTTP ok: HTTP/1.1 200 OK - 0.003 second response time
	SSH	OK	06-22-2005 11:15:38	22d 16h 30m 58s	1/3	SSH OK - OpenSSH_3.1p1 (protocol 1.99)
tcd-gridstore	GFTP	OK	06-08-2005 21:22:57	49d 20h 41m 0s	1/3	TCP OK - 0.236 second response time on port 2811
	SSH	OK	06-22-2005 11:15:38	54d 17h 29m 15s	1/3	SSH OK - OpenSSH_3.6.1p2-CERN20030917 (protocol 1.99)
tcd-gridui	HTTPS	OK	06-22-2005 11:17:37	0d 21h 6m 42s	1/3	HTTP ok: HTTP/1.1 200 OK - 0.071 second response time
	SSH	OK	06-22-2005 11:15:38	54d 17h 31m 56s	1/3	SSH OK - OpenSSH_3.6.1p2-CERN20030917 (protocol 1.99)

FIG. 7.2. *Detailed service monitoring information*

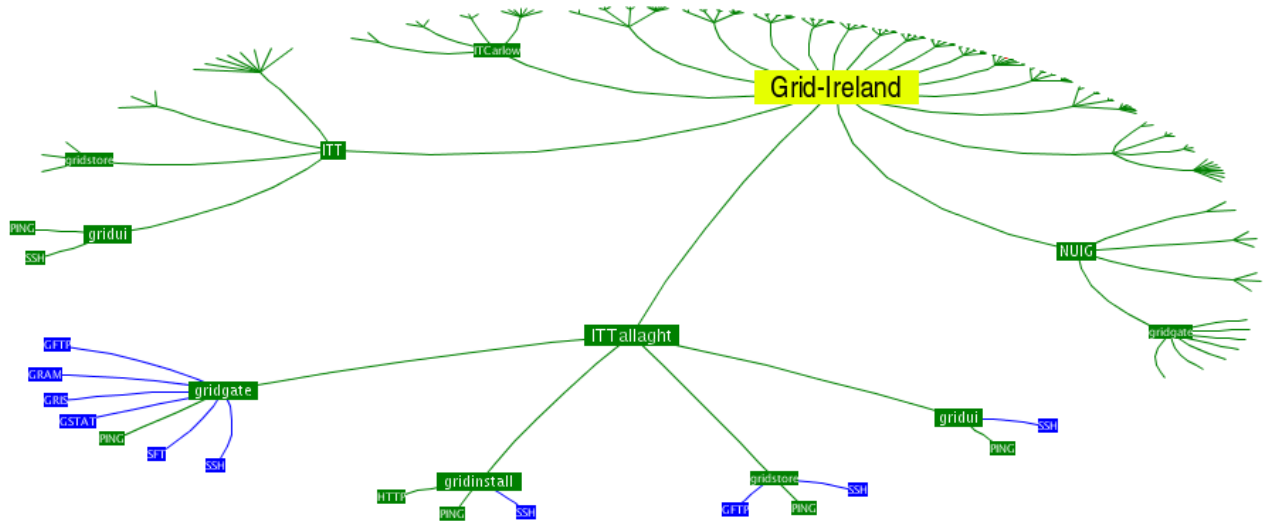


FIG. 7.3. *A snapshot of the Hypergraph Display of Infrastructure Status in mid-navigation*

TABLE 7.1
HyperGraph Infrastructure Monitoring Key

Colour	**Connotation**
Green	No problems or warnings
Amber	Warnings exist for this host or service
Red	Errors or critical warnings exist
Blue	The information for this host is considered stale
Grey	The status could not be determined

7.3. Alert Analysers. Of course, alerts are themselves an interesting data set, amenable to extraction of useful information regarding stability, behaviour, etc. More in-depth analysis and intelligent alerting is made possible through the publication and aggregation of the monitoring information into the R-GMA system [19] and the use of R-GMA consumers in the form of custom alert analysers [16]. These analysers can trigger event handlers and notification mechanisms based on queries made on the R-GMA producers.

It is important to recognise that the power of the SQL queries within individual analysers may be extended by building trees of or a series of dependent analysers, allowing complex analysis of the monitoring information. Analysers may contain advanced processing logic and include historical analysis, providing alert escalation mechanisms or the investigation of correlations among service outages and higher level alerts or security events etc.

Examples of event handlers and notification mechanisms currently under development are outlined in table 7.2.

TABLE 7.2
Example Analyser event handlers

Handler Name	**Description**
Console Handler	Prints the alert to an open console
JDBC Handler	invokes an insert statement on a JDBC resource
RGMA Handler	invokes an insert statement on an RGMA producer
MAILTO Handler	Sends an email to a user/operator detailing alert
SMS Handler	Sends a Short Message Service (text) to a user/operator
XML-RPC Handler	Invokes a method on an XML-RPC server

8. Deployment. The system described has been deployed across 18 Grid-Ireland sites with the task of monitoring almost 200 services on over 70 hosts. Up to 15,000 service/host check results are reported to the central monitoring service each day. It has achieved its objectives and proves to be a valuable tool for the operations team. Its extensible nature is tested regularly as additional checks are requested.

9. Future Work. While the current system is production-ready and has a stable history, a number of future avenues of research are planned. These include further extension to the agent architecture, improved agent and resource management, on-demand monitoring, and the aggregation of additional information sources, particularly with respect to security information [21].

The area of real-time and historical status information analysis, with relevant warning and alerting systems, is where the majority of our future research lies. Customised presentation systems employing 3D visualisation and virtual instrumentation will also be investigated.

A Sourceforge project has been setup [7] to promote the deployment and further development of the architecture. It is hoped that it will attract interst of other users and developers, particularly those interested in tailoring the solution to their own specific infrastructure requirements.

10. Conclusions. This paper has outlined some of the difficulties associated with the effective monitoring of distributed computing infrastructures and has presented our efforts in the development of a new approach to the execution and mangement of grid service monitoring. Following a brief discussion of distributed versus centralised monitoring, we described some of the difficulties encountered in the deployment of monitoring tools within our own infrastructure and introduced our motivations for an agent-based solution. It has demonstrated the combined use of remote and centralised monitoring mechanisms along with the aggregation of existing information systems in order to satisfy the information requirements of a Grid Operations Centre. A number of presentation and alerting systems currently in use were also described.

Our distributed monitoring solution boasts a number of advantages. The lightweight Java agents are easily deployed at remote sites requiring minimal local configuration. Configuration management is achieved through web interfaces to the configuration database. The use of standard communication protocols has resulted in a reliable system, capable of operating within the constraints of tightly managed network security due to its use of standard ports. Site firewall rules for these ports are not typically subject to change following security audits. Where appropriate, existing tools have been used in a flexible and extensible manner, improving the efficiency of the development effort and the overall usefulness of the system. The archiving and republishing of the monitoring information makes this a valuable component on which to base future work.

REFERENCES

[1] Axis - Apache WebServices Project. `http://ws.apache.org/axis/`
[2] CrossGrid. `http://www.crossgrid.org/`
[3] Enabling Grids for E-sciencE (EGEE). `http://www.eu-egee.org/`
[4] Ganglia project documentation. `http://ganglia.sourceforge.net/docs/`
[5] Grid Operations Centre Database. `http://goc.grid-support.ac.uk/gridsite/gocdb/`
[6] Hypergraph project documentation. `http://hypergraph.sourceforge.net/docs.html`
[7] I4c - Wide Area Network Monitoring. `http://i4c.sourceforge.net/`
[8] Interactive European Grid Project. `http://www.interactive-grid.eu/`
[9] Large Hadron Collider Computing Grid project (LCG). `http://lcg.web.cern.ch/LCG/`
[10] Nagios project documentation. `http://www.nagios.org/docs/`
[11] Site Functionality Tests. `http://goc.grid.sinica.edu.tw/gocwiki/Site_Functional_Tests`
[12] S. Andreozzi, N. De Bortoli, S. Fantinel, A. Ghiselli, G. Tortone, and C. Vistoli. Gridice: a monitoring service for the grid. Proc. Cracow Grid Workshop, Poland, December December, 2003.
[13] P. P. Bonnassieux F., Harakaly R. Mapcenter: an open grid status visualization tool. In *ISCA 15th International Conference on parallel and distributed computing systems*, Louisville, Kentucky, USA, September September 19-21, 2002.
[14] R. Byrom, B. Coghlan, A. Cooke, R. Cordenonsi, L. Cornwall, A. Datta, A. Djaoui, L. Field, S. Fisher, S. Hicks, S. Kenny, J. Magowan, W. Nutt, D. O'Callaghan, M. Oever, N. Podhorszki, J. Ryan, M. Soni, P. Taylor, A. Wilson, and X. Zhu. R-gma: A relational grid information and monitoring system. Proc. Cracow Grid Workshop, Poland, December December, 2002.
[15] S. Childs, B. Coghlan, D. O'Callaghan, G. Quigley, and J. Walsh. A single-computer grid gateway using virtual machines. In *Proc. AINA 2005*, pages 761–770, Taiwan, March 2005. IEEE Computer Society.
[16] B. Coghlan and S. Kenny. Grid-wide intrusion detection: First results after deployment. *Submitted to special issue of the Journal of Parallel and Distributed Computing on Security in Grid and Distributed Systems.*
[17] B. Coghlan, J. Walsh, and D. O'Callaghan. Grid-ireland deployment architecture. In P. M. Sloot, A. G. Hoekstra,

T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005*, LNCS3470, Amsterdam, The Netherlands, February February, 2005. Springer.

[18] B. Coghlan, J. Walsh, G. Quigley, D. O.Callaghan, S. Childs, , and E. Kenny. Transactional grid deployment. In M. Bubak, M. Turala, and K. Wiatr, editors, *Proc. Crakow Grid Workshop (CGW'04)*, pages 363–370, Cracow, Poland, December 2004. Academic Computer Centre CYFRONET AGH.

[19] A. Cooke, A. Gray, L. Ma, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Byrom, L. Field, S. Hicks, J. Leake, M. Soni, A. Wilson, R. Cordenonsi, L. Cornwall, A. Djaoui, S. Fisher, N. Podhorszki, B. Coghlan, S. Kenny, and D. O'Callaghan. R-gma: An information integration system for grid monitoring. In *Proc.Int.Conf. Cooperative Information Systems (CoopIS'03)*, Catania, Sicily, November November, 2003.

[20] E. Kenny, B. Coghlan, J. Walsh, S. Childs, D. O'Callaghan, and G. Quigley. Autobuilding multiple ports of computing nodes for grid computing. In *Cracow Grid Workshop (CGW'05)*, Cracow, Poland, November 2005.

[21] S. Kenny and B. Coghlan. Towards a grid-wide intrusion detection system. In P. M. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005*, LNCS3470, Amsterdam, The Netherlands, February February, 2005. Springer.

[22] S. Zanikolas and R. Sakellariou. A taxonomy of grid monitoring systems. *Future Gener. Comput. Syst.*, 21(1):163–188, 2005.

Edited by: Dana Petcu
Received: May 31, 2007
Accepted: June 18, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 291–300. http://www.scpe.org
ISSN 1895-1767

DHR-TREES: A DISTRIBUTED MULTIDIMENSIONAL INDEXING STRUCTURE FOR P2P SYSTEMS

XINFA WEI AND KAORU SEZAKI*

Abstract. Supporting range query over Peer-to-Peer systems has attracted many research efforts in recent years. In this paper, we propose a new multidimensional indexing structure for P2P systems called Distributed Hilbert R-trees (DHR-trees). DHR-trees enables multidimensional range query to be executed similarly as in overlapping regions tree in P2P systems. Its distributed structure makes it fault-tolerant and scalable to dynamic network environment with a large number of peers as well. Our experiments shows that it performs well on multidimensional range query while the maintenance cost is reasonably low.

Key words. peer-to-peer systems, multidimensional queries, Hilbert rtrees

1. Introduction. Enabling efficient access to distributed and dynamic data is hot research topic in P2P systems. Complex queries, such as range queries and k-nearest neighbors queries over multidimensional data, are becoming more important as large amount of data are shared between thousands of user applications. For example, a P2P auction network [17] where peers store information on local real estate (geographical location, price, etc) needs to frequently deal with queries such as "find all real-estate advertisement for properties in a given region in a city". Another query example is that P2P-based sensor data provision networks for environmental surveillance needs to be able to answer request such as "find the average temperature and dioxide concentration within 100 meters from point A".

Multidimensional indexing and range queries problem has been extensively studied in world of the databases. However, in P2P context, this problem brings more challenges because it requires both efficient network-based query processing, fault-tolerant network topology and low cost of network maintenance. Without any central index server, all shared data must be reachable and searchable with the network constantly changing. Each peer node, after receiving query request, should be able to initiate query and decompose query into some sub-queries which are forwarded to other peer nodes when necessary. Therefore one peer node should keep some links and data distribution information to some other peer nodes in the network. There are two extreme solutions for this problem. One is to let peer node keep nothing about information of other nodes and always flood query across the network. This preliminary method is used in earlier P2P systems and is obviously not practical as the network grows very large. Another extreme is to let every peer node keep and refresh all other nodes information all the time to enable query to be executed efficiently and accurately. But maintaining such information requires huge sum of exchange messages which is not affordable in the environment where changes occur frequently in large network.

In this paper, we propose a decentralized multidimensional indexing structure, called DHR-trees (Distributed Hilbert R-trees), that is capable of supporting range queries efficiently and doing network maintenance at a low cost. In DHR-trees, each peer is assumed to control a data set within a minimum bounding rectangle in the space and each peer can be regarded as a leaf node in a Hilbert R-tree. DHR-trees preserves the feature of overlapping regions techniques as in traditional centralized Hilbert R-trees (or R-trees [8] and its variants) which make it can easily accommodate existing R-trees based algorithms with minor modification. DHR-tree of order d provides $O(\log_d N)$ search cost for equality queries, where N is the number of peer in the system. In addition, we introduce the region maintenance of DHR-trees that addresses dynamism problem of P2P systems where peers may be inserted/deleted frequently. We have done some evaluation test of DHR-trees system under various settings. Experiments are performed with different network size, different types of data distribution etc. By testing various settings, the results proved the correctness of P2P systems and effectiveness of range query algorithms. Comparison results also demonstrates that its query execution requires much less node visits than in Squid [14] P2P system.

The rest of this paper is organized as follows: in Section 2, related works are introduced. We then describe in Section 3 DHR-trees structures and the implementation issues. In Section 4, simulation results and comparison result are shown. Finally, conclusion is presented in Section 5.

*The University of Tokyo, 4-6-1 Komaba, Meguro, Tokyo, 153-8505 Japan, email: wei@mcl.iis.u-tokyo.ac.jp, sezaki@iis.u-tokyo.ac.jp

2. Related Work. Many P2P systems have emerged in recent years. Earlier ones are mainly for file sharing applications such as Gnutella [1] and Kazza [2]. They search requested files by flooding messages across network. Unbounded cost causes heavy traffic overhead. Later structured P2P systems, such as Chord [16], Pastry [13], CAN [12] and Tapestry can efficiently support key-based routing and equality searching with logarithmic routing cost. However, the hash function used by these systems destroys the order in the key value space, making them impractical for processing range queries. Recent systems, such as P-Tree [5], P-Grid [3], Skip Graphs [4] and BATON [9], can support one dimensional range queries.

Application with multidimensional data are very important in practical world. Aforementioned systems can not deal with multidimensional data. In contrast, there are some multidimensional index structure solutions available in the centralized database world. However, they are not directly usable in P2P context. Without any central index server, the highly dynamic P2P system must ensure that query can be always answered without missing data. Letting each node maintain links to all other nodes will lead to scalability problem. Ganesan et al. [7] concluded that adapting these solutions to the P2P systems presents four challenges: distribution, dynamism, data evolution and decentralization. Some research efforts has been made and there are progresses in supporting multidimensional range queries. We pick up some most related works and list them as follows:

Mondal et al. [11] proposed P2PR-trees, a variant of R-trees that targets P2P networks. They showed in their simulation study that P2PR-trees show better scalability than two-level Master Client R-trees, since they do not suffer from the central server bottleneck. However, the most important issue in P2P systems, maintenance of a dynamic R-tree is left unaddressed.

The ZNet [15] partitions the data in the native data space in a way as in the generalized quad-tree. ZNet makes use of the Skip Graphs [4] as overlay network routeing in one dimensional space, therefore multidimensional data space is mapped to an one-dimensional index space, such that it can be mapped to nodes in the network. ZNet supports range queries, however, they only provide probabilistic guarantees even when the index is fully consistent due to the underlying Skip Graphs. Moveover, the search performance of the ZNet is $O(d \cdot \log_d N)$ while search performance of DHR-trees is $O(\log_d N)$.

Squid's proposed structure is the closest one to DHR-Trees. In [14], Schmidt C. et al. presents the design of Squid P2P System and its evaluation. The space is first mapped down into an one dimensional space using a Hilbert space filling curve. The one dimensional data is then range partitioned in one dimension and mapped onto the Chord [16] overlay network. For load balancing purpose, however, the original Chord protocol requires that data identifiers are uniformly distributed into one-dimensional space. This restriction makes it inappropriate to directly use location as a data identifier, because data can not always be guaranteed of being distributed uniformly in the space. Furthermore, since routing relies on Chord, which is originally designed for equality search only, the query type is also limited. For example, realizing nearest neighbor query is not easy with Squid since there is no spatial information being preserved.

DHR-Trees also makes use of Hilbert curve to map multiple attributes from m-dimension down into one-dimensional space. But the spatial overlapping region information is kept by individual peers and dynamically maintained. This facilitates multidimensional spatial queries in being executed more efficiently. Unlike other similar approaches, such as Squid and SCRAP [7], DHR-Trees distinguishes itself in supporting R-Tree-like complex queries in dynamic P2P networks.

3. DHR-Trees P2P structure. In multidimensional database, R-Tree [8] and its variants use tree structure for storing information. Each node of an R-tree has a variable number of entries (up to some pre-defined maximum). Each entry within a non-leaf node stores two pieces of data: the child node identifier, and the bounding rectangle of all entries within this child node. Thus, a R-Tree can be regarded as an overlapping region (i. e. rectangle) tree. Search algorithms are usually run from root of the tree; using bounding rectangles to decide whether or not to search inside a child node. Therefore, most nodes in the R-Tree are never "visited" during a search. This leads to high efficiency on query execution.

The Hilbert R-Trees, one of the best-performing multidimensional index structures [6] in the R-Trees family, combines the overlapping regions technique of R-Tree with Hilbert space filling curves. It first stores the Hilbert values of the data rectangle centroid in a B+-tree, then enhances each interior B+-tree node with the minimum bounding rectangle of the sub-tree below. This facilitates the insertion and deletion of new objects considerably. Figure 3.1 demonstrates two dimensional space with some data objects and Figure 3.2 shows its corresponding Hilbert R-Tree.

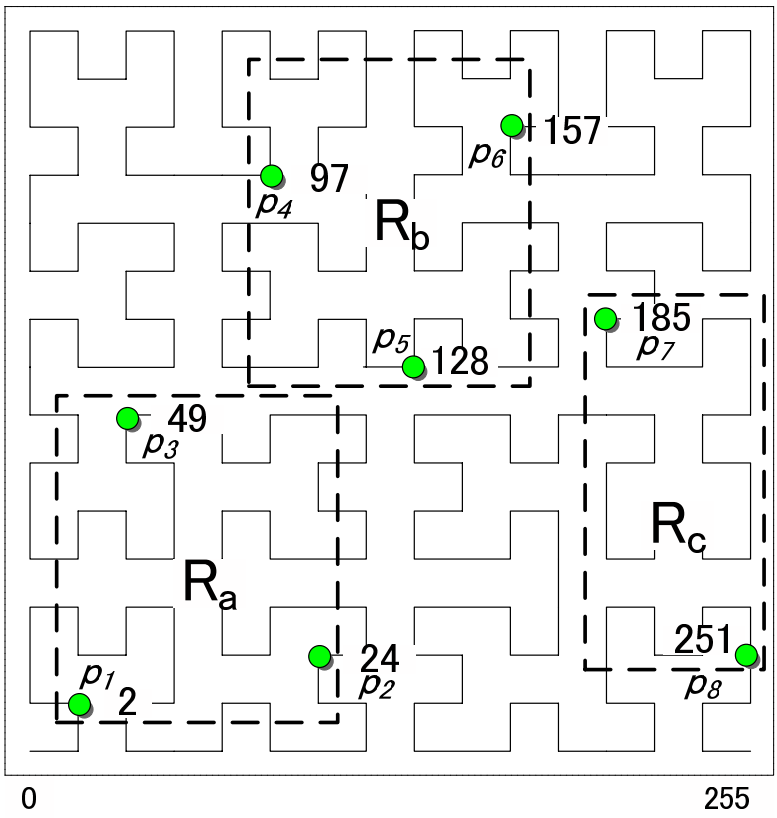


FIG. 3.1. *Data rectangles in a Hilbert R-Tree*

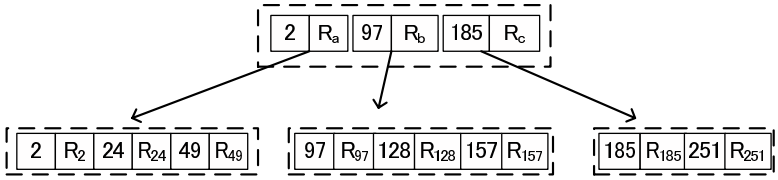


FIG. 3.2. *A Hilbert R-Tree*

P-Trees [5], a recently proposed Peer-to-Peer system, enables one-dimensional range queries to be executed collaboratively by peers. Instead of using Distributed Hash Tables method, it organize all nodes into a closed virtual ring by node identifiers (without hashing). As a result, one-dimensional range queries becomes possible, since the proximity between peers' identifiers is preserved. Each peer independently maintains a routing table, where entries are exactly those on the left-most root-to-leaf path of the B+-Tree. Each peer has its own view of B+-Tree, in which the peer's identifier is located at the left-most leaf node. To keep freshness and correctness of the routing table, peers in the P-Trees system periodically call a stabilization method (called stabilizeLevel).

DHR-Trees combines the advantages of P-Trees and Hilbert R-Trees together. It takes the same topology as in P-Trees. It also enhances peer's routing table with bounding rectangle information. It can be loosely regarded as a distributed version of Hilbert R-Trees, but each node has independent view of the a Hilbert R-Tree. It can also be regarded as a multidimensional extension of P-Trees, enabling indexing and flexibly searching multidimensional information in P2P systems.

3.1. Overview of DHR-Trees. The idea is partially inspired by and based on the work of P-Trees [5], which is designed for supporting one-dimensional range queries in P2P network. The main enhancement is that each peer keeps relevant region tree information in its routing table. Hence it supports the same class of queries as in centralized R-Trees. This is significant since many complex multidimensional query algorithms

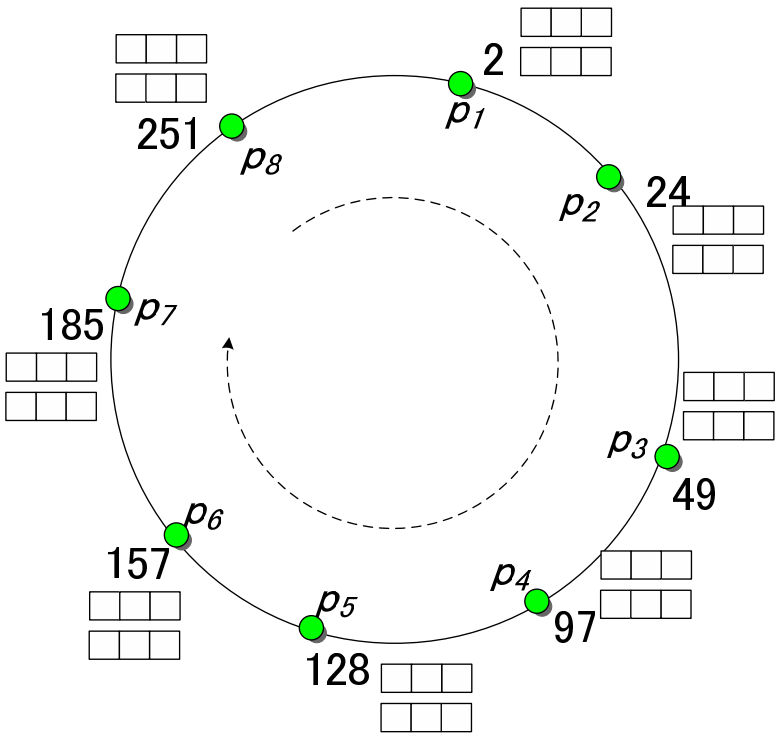


FIG. 3.3. *Virtual ring of peers with routing tables*

can be adapted into the P2P network environment. It discards the notion of maintaining and sharing a globally consistent R-Tree by all peers, and instead maintains semi-independent DHR-Trees at each peer. This allows for fully distributed index maintenance without any need for inherently centralized and unscalable techniques such as primary copy replication.

In a DHR-Trees P2P system, each peer is assumed to control some data set comprised by a Minimum Bounding Rectangle (MBR) in the multidimensional space. Peer p is associated with data rectangle $p.MBR$ and with $p.HCode$, where $p.HCode$ is the Hilbert value of centroid of $p.MBR$ from the space. The $p.HCode$ is used as the index value (key) of p in the one-dimensional space. At the underlying network layer, each peer identifies itself by $p.NetAddress$, which is usually the IP network address. Peers organize themselves into a virtual ring as in P-Tree by their $p.HCode$. The Hilbert curve therefore becomes a closed ring and peers are indexed by their $p.HCode$ in the ring.

Figure 3.3 demonstrates the structure of DHR-Trees for the two-dimensional space case as shown in Figure 3.1. For simplicity, each is assumed to store one point data at its location. By mapping from two-dimensional to one-dimensional space, each peer has a unique Hilbert value. These values are used as peer identifiers in the DHR-Trees P2P system. Peers organize themselves into a virtual ring by their identifiers. Each peer has a pair composed of the predecessor peer and the successor peer as in Chord [16] system. Each peer also has its own composite routing table. The composite routing table is the most important component of DHR-Trees. With Hilbert value and coverage information in the routing table, DHR-Trees P2P system supports routing and equality search similarly as in P-Trees. By having spatial region information in the composite routing table, DHR-Trees P2P system supports multidimensional query directly and efficiently as in centralized R-Trees.

3.2. Composite Routing Table. In DHR-Trees systems, each peer stores nodes information of the left-most root-to-leaf path of independent Hilbert R-Trees from its view in its composite routing table. The significant enhancement to P-Tree is that the region information MBR (we inherit the word MBR from the database world for convenience) is stored as part of the multidimensional overlapping regions tree. The MBR is a minimum bounding rectangle of the sub-tree below (see example in Figure 3.4). As an important part of DHR-Trees, the MBR entry facilitates multidimensional queries, e.g. range queries could be executed efficiently as in centralized R-Trees. Details of the routing table are presented below.

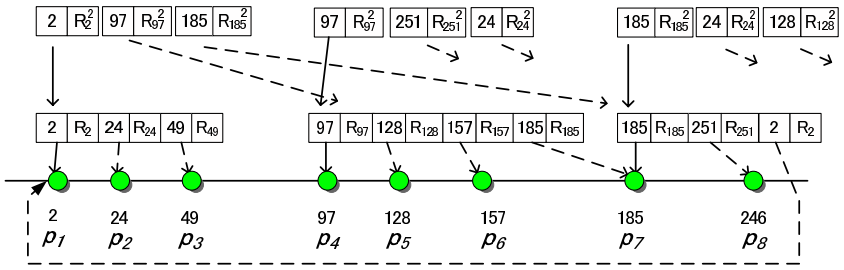


FIG. 3.4. *An example of DHR-Trees with routing tables at p_1 and related peers*

Similar to the P-Tree structure, a peer p maintains a routing table $p.node$, a double indexed array, containing nodes information. Formally, the routing table $p.node$ consists of $numLevels$ rows. There are $p.node[i].numEntries$ of node entries at level i, where $0<i \leq numLevels$. The $numLevels$ is at most $[log_d N]$, where N is the number of peers being indexed and d is the order of the R-Tree. The $p.node[i].numEntries$ is between d and $2d$ at the non-root level of the tree. At root level, $p.node[i].numEntries$ is between 2 and $2d$. Each entry in the row is a triple as following:

$$\langle HCode, MBR, NetAddress \rangle$$

which points to the peer *peer* that owns the MBR and is identified by the index value $HCode$. In the routing table $p.node$, every *peer*'s MBR at level i is a minimum bounding rectangle that comprises all $peer.node[i-1][j].MBR$, where $1 \leq j \leq peer.node[i-1].numEntries$.

Figure 3.4 shows the composite routing table of peer p_1 and related peer for intuitive purpose. The number of levels of p_1, the $p_1.numLevels$, is 2. There are three entries at each level where the first one is about p_1 itself. At level 1, there are 3 entries corresponding to the triples (2,R_2,p_1), (24,R_{24},p_2) and (49,R_{49},p_3) (Due to space limitation, the $NetAddress$ such as p_1 and p_2 etc. is not shown). R_2 is p_1's data MBR. R_{24} and R_{49} are acquired from p_2 and p_3 respectively during p_1 stabilization process on level 1. Notice that p_1,p_2 and p_3 in the triples represent the network address of these peers respectively. At level 2, there are three entries corresponding to the triples (2,R_2^2,p_1), (97,R_{97}^2,p_4) and (185,R_{185}^2,p_7). The MBR at level above 1 is tagged with level i as superscript, such as R_2^2, R_{97}^2 and R_{185}^2. R_2^2 is calculated and updated by calling Algorithm 2. The value of R_{97}^2 and R_{185}^2 are updated at their corresponding peers, p_4 and p_7, and in turn these values are obtained by p_1 during p_1's later stabilization process on level 2. We notice level 1 consists of the closest succeeding peers. At level 2, entries are succeeding peers with *widened* intervals. The interval is equivalent to the coverage range in P-Tree. We can see that these rectangles can be regarded as the left-most route-to-leaf part of the "sliding" Hilbert R-Trees, which has p_1 as its left-most leaf node.

Algorithm 2: p.UpdateMBR (int i)

```
Input: i level of route table, i >1
begin
    reset p.node[i][1].MBR to empty
    j = 1
    while j <p.node[i-1].numEntries do
        p.node[i][1].MBR = CombineRect(p.node[i][1].MBR, p.node[i − 1][j].MBR)
        j++
end
```

3.3. Range Queries. From introduction in previous subsection, it is clear that the rectangle-based overlapping regions tree is maintained in a distributed fashion among peers. This characteristic makes DHR-trees capable of doing R-tree-like range query in P2P systems. Due to the similarity of point query and range query, we focus on how range query is executed.

THEOREM 3.1. *Given the DHR-trees in consistency state, range query initiated at any peer can finally be answered with all qualified data object that exist in the network with guarantee.*

Algorithm 3: p.WindowQuery (Rectangle w, int l)

```
Input: w query window
Input: l level of search to start
begin
    j = 1
    localIntersect = false
    if l >1 then
        while j <p.node[l].numEntries do
            if p.node[i][j].MBR intersects w and l >1 then
                p' = p.node[i][j].peer
                p'.WindowQuery(w, l − 1)
                if j = 1 then
                    localIntersect = true
            j++
    if l = 1 then
        while j <p.node[l].numEntries do
            if p.node[i][j].MBR intersects w then
                p' = p.node[i][j].peer
                p'.searchLocalData(w)
            j++
    if localIntersect then
        this.WindowQuery(w, l − 1)
end
```

In DHR-trees, each peer is ordered by a one dimensional value - Hilbert code and all peers are orderly indexed in the ring of underlying P-Tree. When the P-Tree reached consistent state, any two neighboring entries at same level in route table are guaranteed by P-Tree with coverage property, that is, no peer "missed" by the index structure. Therefore, the coverage property ensures that all peer entries at lower level are contained by entries at higher level. Since each MBR is a combination of all lower *peer.MBR* and the $UpdateMBR$ following level stabilization that works bottom-up from the lowest level to highest level, we can prove all MBR at higher level also comprise all lower MBR by induction bottom-up. Therefore any *peer.MBR* is contained by root level entries. As a result, DHR-tree can guarantee all qualified data object for a query being returned. The pseudo-code of range query process is listed in Algorithm 3. Notice that the result, usually being composed of several pieces, is returned in a asynchronous fashion due to different path length and network latency. The peer initiated query has to be responsible for collecting and synthesizing.

3.4. Wrapping-around problem. Since all peer nodes are indexed in a Hilbert curve ring, a problem about MBR arises: when a peer is near the end of the curve, the MBR entries (i. e. *p.node[i][0].MBR*s) at the upper level of the routing table will need to cover a group of *peer.MBR* located near the start of the Hilbert curve, which is distant from *p.MBR* in the m-dimension space. Intuitively, redundant space will be possibly comprised and it will result in increasing search cost. To solve this problem, we introduce an auxiliary MBR, denoted as *AuxMBR*, for containing the wrapped-around rectangles, while MBR still contains non-wrapped ones. Therefore a more general structure of a route entry should be

$$\langle HCode, MBR, AuxMBR, NetAddress \rangle$$

A null value of *AuxMBR* is allowed when no wrapping-around MBR exists. Accordingly, the range query search (Algorithm 3) should be also altered. Experiment results illustrated in Figure 4.6 shows that the routing cost decreases when this solution is applied.

4. Evaluation. In this section, we present some experimental results we conducted experiments by a implementation of DHR-trees. All our experiment software was implemented in Java and based on PlanetSim [10].

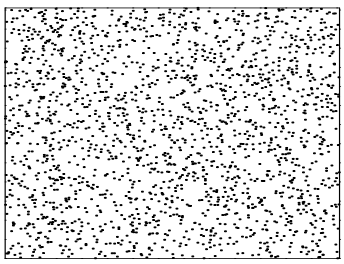


FIG. 4.1. *Uniform*

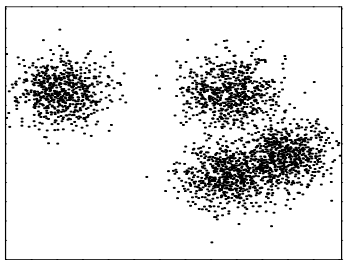


FIG. 4.2. *Clustered*

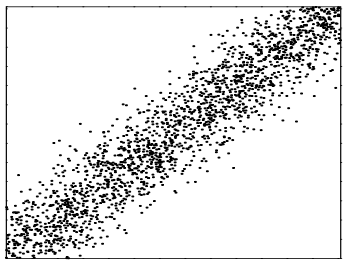


FIG. 4.3. *Skewed*

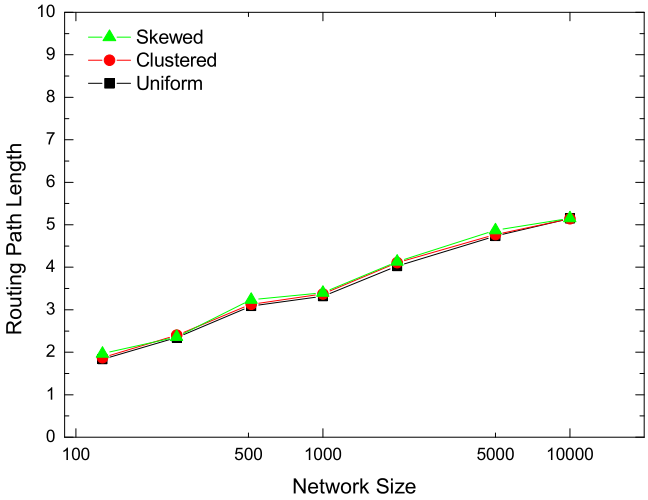


FIG. 4.4. *Routing Cost*

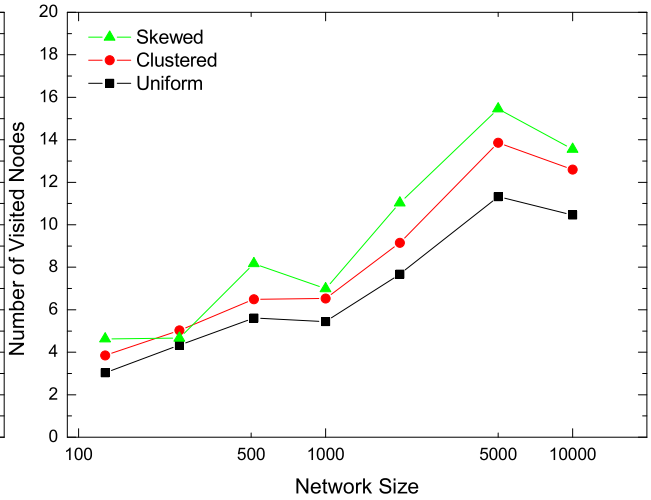


FIG. 4.5. *Point query cost*

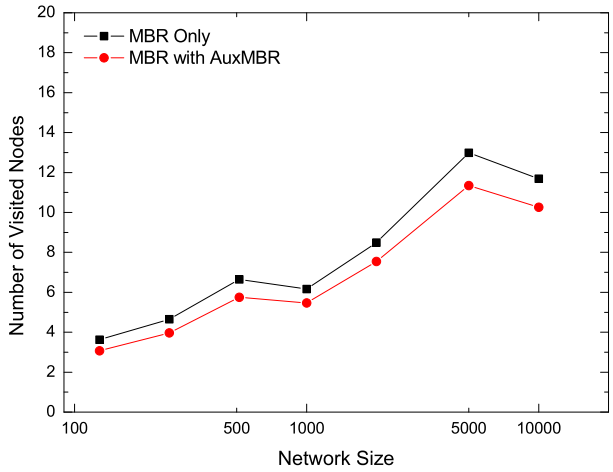


FIG. 4.6. *Improvement with Auxiliary MBR*

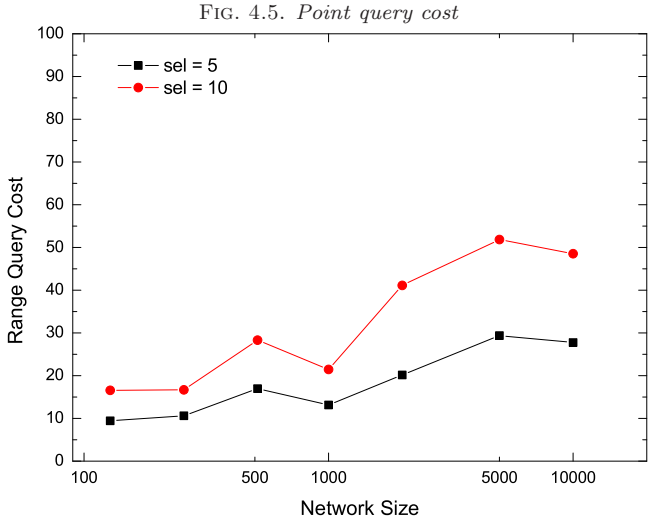
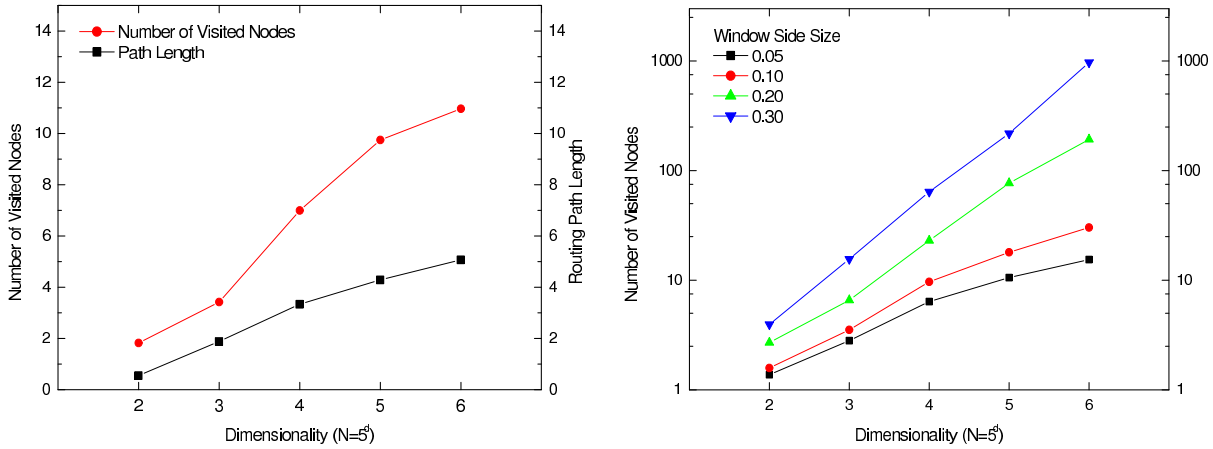


FIG. 4.7. *Range Queries*

All experiments were performed on a machine (3.0GHz Intel Pentium ET) with 2GBytes of main memory, running Windows XP Professional. In our current simulation environment , the DHR-trees nodes were configured to run in a single Java Virtual Machine.

4.1. Two-dimensional DHR-trees. In our simulations, we first use a two dimensional space. This space is mapped by a Hilbert curve of order 16 down into a one dimensional space such that it is partitioned into a $2^{16} \times 2^{16}$ grid. For simplicity of evaluation, each peer is assumed controlling a rectangle of size 1×1. The order d of R-tree is fixed as 4 and network size varies from 128 to 10000. Simulation program begins with building DHR-trees with configuration as the initial phase. Node joins into DHR-trees every several steps in a random order and stabilization process of P-Trees works periodically together with $UpdateMBR$ process.



(a) Routing path length and Query cost (b) Range queries

FIG. 4.8. *Multidimensional DHR-trees*

Insertion/deletion, routing and query experiments are executed after the network becomes stabilized. Three type of distribution of data sets are used: uniform, clustered and skewed distributions as shown in Figure 4.1, Figure 4.2, Figure 4.3 (Each with 2000 points in Figures).

Point Query Cost. Poinst query cost is measured by number of visited nodes during query. The different distribution incurs difference on number of visited nodes as shown in Figure 4.5: routing in case of skewed distribution visited more number of nodes. This is because that skewed data set make more extensive overlapping MBR in route table so that routing are more frequently forwarded to some unnecessary nodes.

Improvement by Auxiliary MBR. In Section 3.4, the auxiliary MBR is introduced to solve extensive coverage problem of wrapping-around MBR. Experiment in Figure 4.6 (uniform distribution) show that this approach reduces about $12\% \sim 13\%$ routing messages in case of uniform distribution.

Range Queries. In range query experiments, we use data set in uniform distribution, where size N varies from $128 \sim 10000$. We use rectangles as query window with side length to be adjusted in accordance with network size, such that *sel* number of result peers (MBRs) could be expected to fall into query window. In our experiments, *sel* specified as 5 and 10 separately. The location of query window is randomly selected and peers that initiate query are also randomly picked from the network. Query cost are measured by number of visited nodes. For each combination of N and *sel*, 1000 times queries are executed and results are averaged. The result shown in Figure 4.7 indicates that range query can be efficiently executed in DHR-trees.

4.2. High Dimensional DHR-trees. Some experiments with high dimensional data has also been conducted. Space dimensionality m is changed from 2 to 6. Network size is set as 5^m, changing from 25 ($m = 2$) to 15625 ($m = 6$). In all dimensionality configuration, side length of space fixed at 1024 and Hilbert curve of order 10 is used. Multidimensional data is distributed uniformly in the space. Auxiliary MBR is enabled in all experiments.

Routing Path and Number of Visited Nodes. In Figure 4.8(a), length of routing path increases slowly as d and N increases. This indicates that basic routing does not require much hops even in multidimensional spaces. The number of visited nodes, i. e. point query cost, increases with network size and is always bigger than routing path like in 2-dimensional case.

Range Queries. Three group of range query experiments has been executed. In three groups, query window size is set to be 0.05, 0.1 and 0.2 respectively. As for combination of query window size and dimensionality m, the center of query window is randomly chosen in space and query is executed 1000 times to average results. Figure 4.8(b) shows: With small windows size, query cost increases slowly. With big window size, however, the query cost increases rapidly with dimensionality and network size. This can be explained as that bigger query window overlaps much more MBRs in DHR-trees in multidimensional space and more nodes will have to be involved in query.

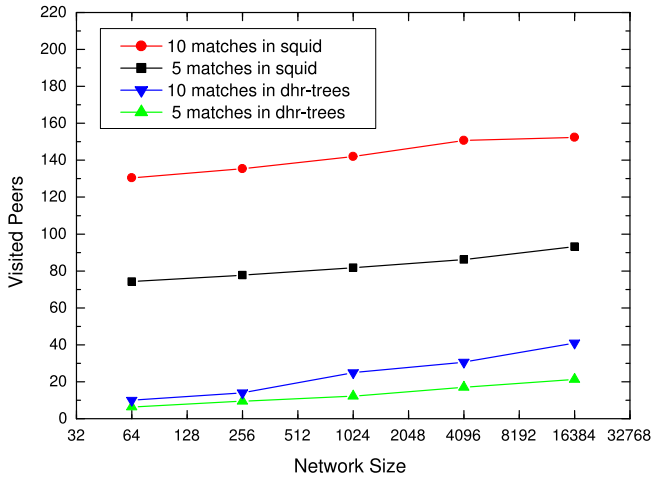


FIG. 4.9. *Query performance comparison with Squid*

4.3. Performance Comparison with Squid. As introduced in Section 2, Squid P2P system is the most related to DHR-Trees as they have the same method of mapping higher dimension space to one-dimensional space and use the same virtual ring as the overlay network structure. The most different part is that it uses a different routing table (called finger table) and a different query execution mechanism. Squid does cluster refinement to decompose query ranges into smaller query clusters recursively until it reaches *finest* level of the Hilbert curve, since top-down queries are not supported as each peer has no region information in its routing tables; In contrast, DHR-Trees supports R-Trees-like queries directly by having MBR region information maintained in peer's routing table. Hence, DHR-Trees visits less peers and generates less traffic messages when performing range queries, proving it is superior to Squid in efficiency of query execution. Figure 4.3 shows the comparison results of range queries. Moreover, nearest neighbor query is even not supported by Squid system because spatial proximity in multidimensional space is not preserved after mapping into one-dimensional space.

5. Summary. In this paper, we proposed a new multidimensional indexing structure called Distributed Hilbert R-trees. DHR-trees enables multidimensional range query to be executed efficiently as in R-trees. Its distributed structures make it fault-tolerant and scalable to a large number of peers. DHR-trees uses Hilbert curve in mapping from m-dimension down into one dimensional key space and uses P-Trees in building overlay network. Similarly as in Hilbert R-trees, this mapping is used just for ordering of each peer in the P-Tree, while inherited overlapping regions technique can efficiently support flexible queries in P2P systems. To the best of our knowledge, DHR-trees is the first P2P systems that utilizes semi-independent R-tree structure and addresses dynamic feature of network as well. Our current experiments shows that it performs well on multidimensional range query and the maintenance cost is reasonably low comparing with previous approaches. As future work, we plan to compare it with other approaches and test it in a real-world application.

REFERENCES

[1] *The gnutella web site.* `http://gnutella.wego.com`
[2] *The kazaa web site.* `http://www.kazaa.com`
[3] K. ABERER, *P-Grid: A Self-Organizing Access Structure for P2P Information Systems*, vol. 2172, Jan. 2001.
[4] J. ASPNES AND G. SHAH, *Skip graphs*, in SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms, Philadelphia, PA, USA, 2003, Society for Industrial and Applied Mathematics, pp. 384–393.
[5] A. CRAINICEANU, P. LINGA, J. GEHRKE, AND J. SHANMUGASUNDARAM, *Querying peer-to-peer networks using p-trees*, in WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, New York, NY, USA, 2004, ACM Press, pp. 25–30.
[6] V. GAEDE AND O. GUNTHER, *Multidimensional access methods*, ACM Comput. Surv., 30 (1998), pp. 170–231.
[7] P. GANESAN, B. YANG, AND H. GARCIA-MOLINA, *One torus to rule them all: multi-dimensional queries in p2p systems*, in WebDB '04: Proceedings of the 7th International Workshop on the Web and Databases, New York, NY, USA, 2004, ACM Press, pp. 19–24.
[8] A. GUTTMAN, *R-trees: a dynamic index structure for spatial searching*, in SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data, New York, NY, USA, 1984, ACM Press, pp. 47–57.

[9] H. V. Jagadish, B. C. Ooi, and Q. H. Vu, *Baton: a balanced tree structure for peer-to-peer networks*, in VLDB '05: Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, pp. 661–672.
[10] H. T. N. Jordi Pujol Ahullo, Ruben Mondejar Andreu, *Planetsim.* `http://planet.urv.es/planetsim/`
[11] A. Mondal, Y. Lifu, and M. Kitsuregawa, *P2PR-Tree: An R-Tree-Based Spatial Index for Peer-to-Peer Environments*, vol. 3268, Jan. 2004.
[12] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker, *A scalable content-addressable network*, in SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2001, ACM Press, pp. 161–172.
[13] A. Rowstron and P. Druschel, *Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems*, vol. 2218, Jan. 2001.
[14] C. Schmidt and M. Parashar, *Enabling flexible queries with guarantees in p2p systems*, Internet Computing, IEEE, 8 (2004), pp. 19–26.
[15] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou, *Supporting multi-dimensional range queries in peer-to-peer systems*, in Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on, 2005, pp. 173–180.
[16] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, *Chord: A scalable peer-to-peer lookup service for internet applications*, in SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, New York, NY, USA, 2001, ACM Press, pp. 149–160.
[17] E. Tanin, A. Harwood, and H. Samet, *A distributed quadtree index for peer-to-peer settings*, in Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on, 2005, pp. 254–255.

Edited by: Dana Petcu
Received: May 29, 2007
Accepted: June 19, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 301–311. http://www.scpe.org
ISSN 1895-1767

SENSITIVITY ANALYSIS OF WORKFLOW SCHEDULING ON GRID SYSTEMS*

MARíA M. LÓPEZ, ELISA HEYMANN, MIQUEL A. SENAR†

Abstract. Scheduling is an important factor for the efficient execution of computational workflows on Grid environments. A large number of static scheduling heuristics has been presented in the literature. These algorithms allocate tasks before job execution starts and assume a precise knowledge of timing information, which may be difficult to obtain in general. To overcome this limitation of static strategies, dynamic scheduling strategies may be needed for a changing environment such as the Grid. While they incur run-time overheads, they may better adapt to timing changes during job execution. In this work, we analyse five well-known heuristics (min-min, max-min, sufferage, HEFT and random) when used as static and dynamic scheduling strategies in a grid environment in which computing resources exhibit congruent performance differences. The analysis shows that non-list based heuristics are more sensitive than list-based heuristics to inaccuracies in timing information. Static list-based heuristics perform well in the presence of low or moderate inaccuracies. Dynamic versions of these heuristics may be needed only in environments where high inaccuracies are observed. Our analysis also shows that list-based heuristics significantly outperform non-list based heuristics in all cases and, therefore, constitute the most suitable strategies by which to schedule workflows either statically or dynamically.

Key words. workflows, scheduling, grid

1. Introduction. Grid environments connect distributed and heterogeneous resources in a seamless system that involves coordinating and sharing computing, application, data, storage, or network resources across dynamic and geographically dispersed organizations [1]. Grid systems offer high computing capabilities that are used in many scientific research fields. Biologists, chemists, physicists and other researchers have therefore become intensive users of applications with high performance computing characteristics. Several projects such as GriPhyn [2], DataGrid [3], GridLab [4], GrADS [5] or CrossGrid [6], whose middleware infrastructure is intended to simplify application deployment on computational grids, have also developed special middleware for defining and managing scientific workflows.

Scientific workflows are concerned with the automation of scientific processes that consist of inter-dependent jobs, where tasks are structured on the basis of their control and data dependencies. Workflow applications are an important class of programs that can take advantage of Grid-system power, as has been shown in the LIGO [7] pulsar search, several image processing applications [8] or the ATLAS physics experiment [9].

Any workflow management system requires certain key elements:

1. Definition and composition of workflow components.
2. Task mapping and scheduling during execution.
3. Data movement between dependent tasks.
4. Handling of execution failures.

From the above list of issues, we focus on the second, which is an ongoing research effort covered by much recent work. In general, workflow scheduling strategies cover different trade-offs between dynamicity and look-ahead. On the one hand, decisions about the resource on which to run a particular task can be made as soon as possible (early) or just before the task is executed (in-time). On the other hand, decisions can be made with reference to information about the whole workflow or simply the task at hand. Obviously, the complexity and run-time overhead incurred in the decision-making process increases as global information and in-time strategies are used. The design of a scheduling strategy also depends on what objective function the user wants to minimize or maximize (e.g. minimizing overall job completion time or maximizing resource utilization). Our objective function is to minimize overall job completion time or the application makespan. The makespan of a workflow application is defined as the time at which the last component of the workflow finishes execution.

Traditionally, scheduling strategies assume the existence of deterministic information about processing times for all workflow tasks and for their corresponding communications. These times may be provided by the user or obtained by profiling techniques, and are generated before the execution. Current information on the execution environment can also be obtained by on-line tools, such as NWS [10] and is needed to make scheduling decisions at run-time. Unfortunately, accurate models of processing and communication times are very hard

*This work was made in the frame of the "int-eu.grid" project (sponsored by the European Union), and supported by the MEyC-Spain under contract TIN 2004-03388, and partially supported by the NATO under contract EST.EAP.CLG 981032

†Departament d' Arquitectura d' Ordinadors i Sistemes Operatius Universitat Autònoma de Barcelona, Barcelona, Spain mmar@aomail.uab.es, {elisa.heymann, miquelangel.senar}@uab.es

to obtain in practice and, therefore, schedules based on computation and communication estimates may suffer from significant timing changes at run-time.

Many static heuristics have been proposed in the literature. Heuristics based on list scheduling are among those that provide good quality schedules at a reasonable cost. These strategies [11, 12, 13] schedule the tasks in the order of their previously computed priority. Thus, at each scheduling step, a task is first selected and its processor is then calculated. HEFT [11] is considered to be one of the best strategies within the group of list scheduling heuristics. However, in [13] it was observed that its performance is affected by the approach followed to assign weights to the nodes and edges of the graph. A recent study [14] has also analyzed the performance of a low-cost rescheduling policy, which attempts to reschedule tasks selectively during workflow execution (hence, without incurring a high overhead).

Other studies have analyzed another set of heuristics that do not take into account the whole structure of workflow dependencies during the scheduling process. The graph is analyzed according to the dependencies but each task does not have a predefined priority. So, at each scheduling step, each ready task is tentatively assigned to every processor and the best (task, processor) pair is selected. Thus, at each step, both a task and its corresponding processor are selected at the same time. Different strategies use different criteria to select the best pair (for instance, the pair that guarantees the minimal completion time for all ready tasks, the pair that provides a minimum starting time for all ready tasks, etc). Among others, strategies such as min-min [15], max-min [16] and sufferage [17] have been used and analyzed in recent work [18, 19].

In general, list-scheduling strategies exhibit a higher computational complexity than strategies from the other group. They perform well for irregular and communication-intensive graphs, where the scheduling of crucial tasks first seems to be very important. Strategies that are not based on list-scheduling perform well on computation-intensive graphs with a high degree of parallelism, in which a maximum utilization of processors seems to be one of the most important factors to be considered.

However, to the best of our knowledge, no comparative studies have been carried out between list-scheduling strategies and non-list scheduling strategies. In our work, we have compared strategies from both groups and we study their behavior both when they are applied statically and dynamically.

The remainder of the paper is organized as follows. Section 2 provides some background and describes the assumptions that we consider within our problem setting. Section 3 illustrates the problem of timing inaccuracies on static scheduling heuristics. In section 4, we analyze the sensitivity of several heuristics to timing changes during graph execution and also study the benefits of dynamic rescheduling applied to all the strategies. Finally, section 5 concludes the paper.

2. Background. Many complex applications consist of inter-dependent jobs that cooperate in order to solve a particular problem. The completion of a particular job is the condition needed to start the execution of those jobs that depend upon it. This kind of application workflow may be represented in the form of a DAG a directed acyclic graph. A DAG is a graph with one-way edges that does not contains cycles. DAGs are frequently used to represent a set of programs where the input, output, or execution of one or more programs is dependent on one or more other programs. The programs are nodes (vertices) in the graph, and the edges (arcs) identify the dependencies of these programs. Fig. 2.1 represents an example DAG corresponding to the MB application [20].

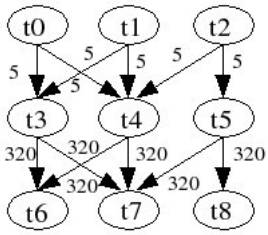


FIG. 2.1. *DAG corresponding to the MB application.*

Associated to each node n and machine m, there is the real execution time, which represents the exact amount of time that takes to execute node n on machine m. In practice, in a real grid environment, real execution times (corresponding to both task computation and intertask communications) are not available;

however we use them for comparative purposes. Fig. 2.2(b) shows the real execution times for the MB nodes when executed on 3 different machines (M1, M2 and M3).

Machines	Transfer Rate real (Mbits per sec)	Transfer Rate estimated (Mbits per sec)
M1-M2	11	15
M1-M3	11	13
M2-M3	8	5

(a)

	Real Execution Time			Estimated Exec. Time		
	M1	M2	M3	M1	M2	M3
t0	227	232	309	265	269	339
t1	168	172	230	197	168	202
t2	150	153	205	174	136	173
t3	299	306	408	338	272	433
t4	311	318	424	275	343	437
t5	299	306	408	341	278	439
t6	281	288	384	255	246	335
t7	362	370	494	361	333	476
t8	193	197	264	184	222	294

(b) (c)

FIG. 2.2. *(a) Communication time between machines (real and estimated) . (b) Real execution times in seconds. (c) Estimated execution times (variation=50%).*

As real execution times are in practice not available at submission time, in our work we assume that estimates of the execution times of the nodes are known in advance. We calculate these estimated by varying the real value randomly by a certain percentage. Fig. 2.2 (c) shows an example of varying the execution times of Fig. 2.2 (b) up to $\pm 50\%$.

In addition to the nodes computation time, there is a communication volume associated to each pair of communicating nodes. These times in Mbytes are shown beside each edge on Fig. 2.1.

Finally there is a communication transfer rate associated to each pair of machines, which are shown in Fig. 2.2(a). In this example, differences between real and estimated transfer rates have been computed by applying a 50% variation to real transfer rates.

Grid environments are composed of a large number of heterogeneous machines. Commonly, users have an estimated knowledge of their DAG timing when it is executed on their local machines, but we have no guarantees about the accuracy of this estimated execution time (as the DAGs jobs may be executed with different input data, on different machines and so on). In the following section, we illustrate the problems derived from this fact when scheduling workflows.

3. Scheduling Sensitivity to Timing Inaccuracies. We now show an example of the makespan obtained when scheduling the MB DAG described in the previous section.

Usually only estimated information about both the jobs' execution and communication time is available. By scheduling this DAG using a particular scheduling strategy (HEFT) applied to the estimated execution time of the jobs on 3 machines, the Gantt chart in Fig. 3.1 is obtained. Slashed boxes represent communication before the execution of a node. For the sake of simplicity in this example we asume that communication times are known. It is worth mentioning that in the rest of the paper we consider that exact communication times are not known, as it happens in reality. In the figure we see that tasks 0, 4 and 6 were assigned to machine M1, tasks 1, 5 and 8 to machine M2, and tasks 2, 3 and 7 to machine M3.

But after executing tasks on machines we get the real execution time, which is frequently different to the estimated one. Fig. 2.2 (b) shows a variation of the real execution time up to a 50% of the estimated time. If we consider these inaccurate estimated execution times and apply a machine normalizing factor for the machines M1, M2 and M3, the Gantt chart in Fig. 3.2 is obtained. In this case the tasks are executed on the same machines as in the previos Gantt chart, but we measure the obtained real makespan.

Having accurate information about the nodes execution time is not quite realistic when executing real applications on real environments, but it is useful to see how makespan is worsened when this information is inaccurate. Fig. 3.3 shows the Gantt chart obtained when real execution times are known. We see that the makespan experienced a worsening of 10.3% when the estimated times were considered (Fig. 3.2)

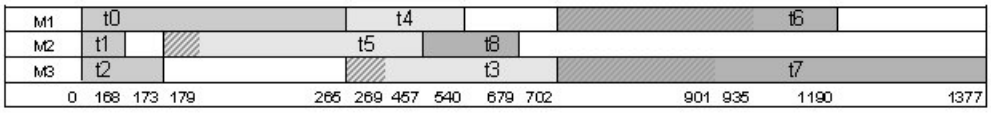


FIG. 3.1. *Gantt chart obtained using the HEFT policy when only estimated execution times are known.*

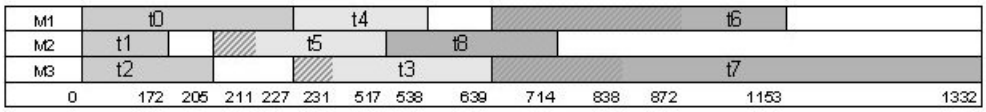


FIG. 3.2. *Gantt chart obtained using the HEFT strategy when estimated execution times are known and real execution times are obtained.*

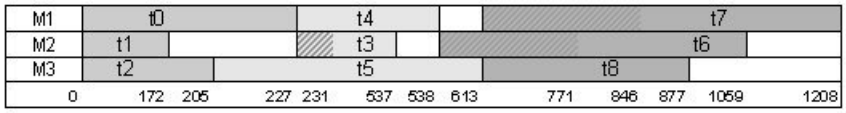


FIG. 3.3. *Gantt chart obtained using the HEFT strategy when real execution times are known.*

As the scheduling strategies for workflows (described in the following section) are sensitive to the accuracy of the estimated execution times, we studied how these strategies behave when performing Rescheduling. Rescheduling consists of updating the value of the estimated execution time for all the pending nodes of a specific machine (the one on which the task that has just finished was executed), according to the execution time value obtained by the recently completed task. Fig. 3.4 depicts the Gantt chart obtained when applying rescheduling to our example DAG. In this case, tasks 0, 5 and 7 were assigned to machine M1, tasks 1, 4, 6 to machine M2, and tasks 2, 3 and 8 to machine M3. It can be seen that the makespan worsening is reduced to 1.16% with respect to the ideal situation in which real execution times of the tasks are known in advance.

In general, rescheduling will not provide the makespan obtained with the real (and therefore, accurate) execution time values, but it provides an improvement on the makespan obtained when only estimated execution times are known. We performed a systematic and exhaustive study with the aim of evaluating the behavior and benefits obtained when rescheduling. This study is presented in the following section.

4. Experimental Study. In this section, we evaluate the performance of several scheduling strategies with respect to the makespan obtained when these are applied to scheduling both fork-join and random workflows on heterogeneous systems. We test various scheduling strategies under different scenarios:

1. Knowing in advance the real execution time of the applications nodes. Clearly this case does not correspond to a real situation, but it is used for comparative purposes.

2. Knowing only an estimate of the nodes execution time and the intertask communication times. This scenario pertains when there is an estimate of task execution times and communication rates, possible obtained from previous executions, which may (or may not) be accurate.

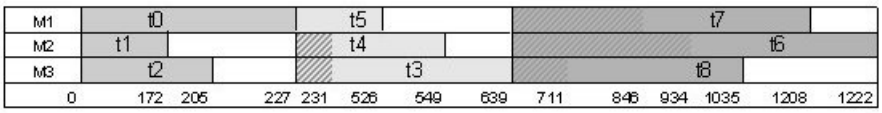


FIG. 3.4. *Gantt chart obtained using the HEFT policy when rescheduling.*

3. Knowing only an estimate of the nodes execution time and the intertask communication times, but being able to dynamically *reschedule* the remaining tasks after each task finishes. In this case, accurate information obtained for a just-executed task is used to update the estimated execution times of the remaining tasks.

As the main result from these simulation experiments, we are interested in obtaining information relating to how rescheduling strategies perform on average, and which scheduling policy is most suitable for performing such rescheduling.

4.1. Heuristics Description. The set of scheduling strategies studied were the following:

1. **Min-min:** For each task Ti, the machine Mi having the minimum estimated completion time for Ti is found, and the tuple (Ti, Mi, ET) is formed, ET being the execution time of Ti on Mi. Of all the tuples, that with minimum ET is scheduled. These steps are repeated iteratively until all the tasks have been scheduled.

2. **Max-min:** For each task Ti, the machine Mi having the minimum estimated completion time for Ti is found, and the tuple (Ti, Mi, ET) is formed, ET being the execution time of Ti on Mi. Of all the tuples, that with maximum ET is scheduled. These steps are repeated iteratively until all the tasks have been scheduled.

3. **Random:** Tasks ready to be executed are assigned randomly to the machines. This strategy represents the case of a pure dynamic method that has no information on the application. In principle, this should obtain the worst performance of all the strategies presented; therefore it will be used as a lower bound in the performance achievable by the other strategies.

4. **Sufferage:** In this strategy, both the minimum and second best minimum ET values are found for each task. The difference between these two values is the sufferage value. The task having the maximum sufferage value is scheduled next. These steps are repeated iteratively until all the tasks have been scheduled.

5. **HEFT:** This strategy selects the task with the highest upward rank value at each step, and assigns the selected task to the processor which minimizes its earliest finish time.

4.2. Simulation Framework. All described scheduling strategies have been systematically simulated for the three possible scenarios, in order to obtain the makespan, considering the following factors:

1. *Number of Machines:* This represents the number of processors on which each DAG is executed. The features of the machines used in the simulation were taken from executing real applications on real machines. We performed simulations using 5, 10, 15 and 20 machines. Machines are heterogeneous, and communications among them were characterized by the bandwidth values obtained experimentally.

2. *DAGs:* This represents the DAGs considered. We generated 100 completely random DAGs, and also 100 fork-join random graphs. The number of nodes ranged from 50 to 250. Thus we examine systems with a medium or large amount of tasks. The generated fork-join graphs contained a number of levels between 5 and 20, and the number of nodes per level ranged between 5 and 25. Task computing-time values ranged between 100 and 500 units, while communication times among tasks were also randomly generated and ranged between 1 and 20 Mbytes per second.

3. *Variation:* This represents the percentage of error that the estimated values may have with respect to real execution times and real communication times. We considered values of 10%, 30%, 50%, 70% and 90%. When a 10% variation was used, task execution times and communication times were fairly reliably estimated. When a 90% variation was used, tasks exhibited significant changes in their execution, corresponding to applications with highly erroneous estimations.

It should be observed that, in our simulations when two adjacent nodes were executed on the same machine, the associated communication cost is 0, while if these two nodes are executed on different machines, the communication cost is weighted up to the bandwidth of the link communicating those machines.

Time variations have been applied in a congruent way. By congruent we mean that in our simulations we assume a set of processors P1, P2, , Pi, with a fixed performance ratio between each pair. As a consequence, when task execution times and their variations were generated, execution times in the fast processor P1 were

always smaller than in the slow processor P2, and their ratio was always the performance ratio between P1 and P2; i. e., if the execution time in processor P1 for tasks T1 and T2 is 10 and 20, respectively, and if processor P2 is twice as fast as P1, then the execution times for tasks T1 and T2 in processor P2 will be 5 and 10, respectively.

Given a number of machines, a DAG and a variation, our simulator computes the makespan obtained for the five strategies studied for the three scenarios previously described (considering exact task execution times, estimated execution times and rescheduling). The particular features of the simulated grid nodes were taken from a real grid environment using the Globus Information Index [21] for machine information, and the Network Weather Service tool [10] for information about the network bandwidth.

Each DAG was simulated using 10 different sets of grid machines (out of 20). In the next subsection we show the average of the makespan obtained considering the 100 DAGs for each scenario.

4.3. Simulation Results. Although we have conducted tests for all the values commented on, in this section we present only those results that are the most interesting. We will illustrate—with figures—the results for 5 and 20 machines since these prove to be sufficiently representative for the results obtained with an intermediate number of machines.

In the rest of the section, certain pertinent result figures for makespan time are presented. The Y-axis contains average makespan. We now review the most relevant results obtained from our simulations.

Fig. 4.1 and Fig. 4.2 show the negative effect of varying the estimated execution time for all the scheduling strategies in the case of 5 (Fig. 4.1) and 20 machines (Fig. 4.2) for an average of 100 random generated graphs. The average makespan is obtained for the 5 strategies (min-min, max-min, sufferage, heft and random) under the cases of real and estimated information on task execution times. The X-axis contains the variation, ranging from low-variation applications (10%) to highly inaccurate estimations (90%).

Simmilarly, Fig. 4.3 and Fig. 4.4 show the results obtained by the same heuristics when real and estimated execution and communication times. In this case, results are slightly worse than in the previous one because errors in communication times are added to errors on execution times.

Fig. 4.5 and Fig. 4.6 shows a reduction of the negative effect of the makespan when rescheduling, considering different variations, for all the scheduling strategies in case of 5 (Fig.4.5) and 20 machines (Fig. 4.6), for an average of 100 random generated graphs. The average makespan is obtained for the 5 strategies (min-min, max-min, sufferage, heft and random) for estimated (i. e., inaccurate) cases and when performing rescheduling. The X-axis contains the variation, ranging from low-variation applications (10%) to highly inaccurate estimations (90%).

Fig. 4.7 and Fig. 4.8 show simulation results obtained when rescheduling was applied to the cases that exhibited variations in execution and communication times.

It is observed that, regardless of the number of machines, if the real times for tasks and communications are not known (the case in which estimated times are used), higher makespans are obtained. As expected, the higher the variation, the higher the execution time worsening for the whole DAG. In particular, when the variation applied to the estimated execution times of the graph nodes is 10%, makespan worsening is close to 5%; in contrast, when the variation is 70%, this worsening increases up to 10%. In general, non-list scheduling strategies are more sensitive than list-based scheduling strategies to inaccuracies in timing information.

In a grid environment, where machines are heterogeneous and the conditions of the system change dynamically, the real execution times of the tasks are usually not known. Therefore, scheduling a DAG through a simple non-list scheduling policy such as min-min or max-min using estimated execution times for the tasks would lead to a poor makespan. This situation is alleviated if we take advantage of the information on the machines current situation, given by recording the execution time for those tasks that have just finished their execution, and by performing rescheduling. In other words, we only have an estimation of the nodes execution time, but are able to dynamically reschedule the remaining tasks after each task finishes.

Static list-based heuristics perform well in the presence of low or moderate inaccuracies in timing information. Nevertheless, dynamic versions of these heuristics are needed when high inaccuracies are detected.

Now that we have determined that rescheduling reduces makespan, especially in cases of high variations for DAG task execution time, the consequent question to address is which scheduling strategy introduces greater benefits when rescheduling. Fig. 4.9 and Fig. 4.10 depicts the average makespan for all the strategies using rescheduling for a 70% variation. The X-axis contains the number of machines, and the Y-axis the average makespan.

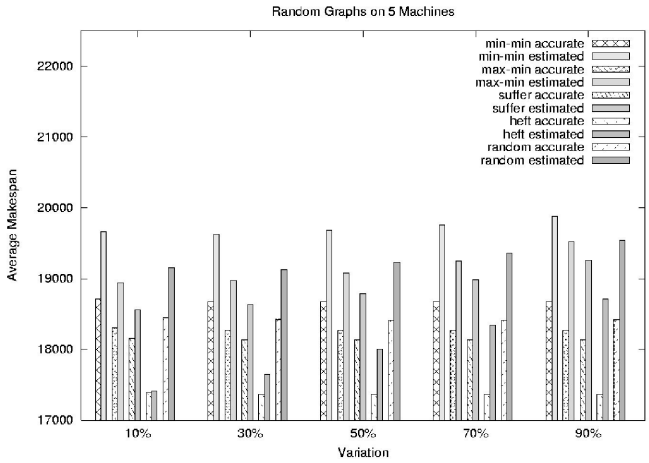


FIG. 4.1. *Effect of the variation of execution time considering 5 machines.*

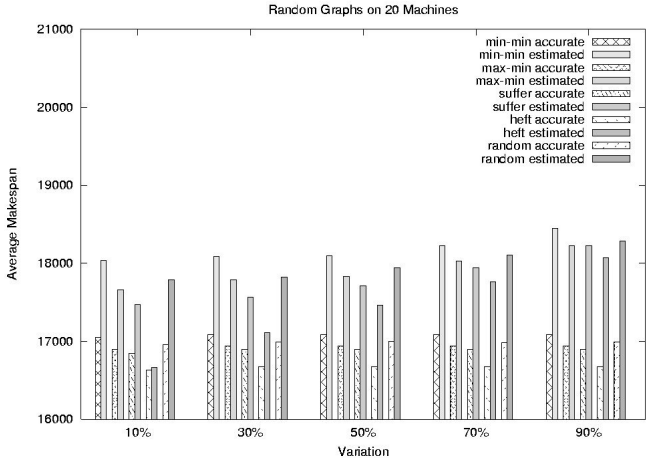


FIG. 4.2. *Effect of the variation of execution time considering 20 machines.*

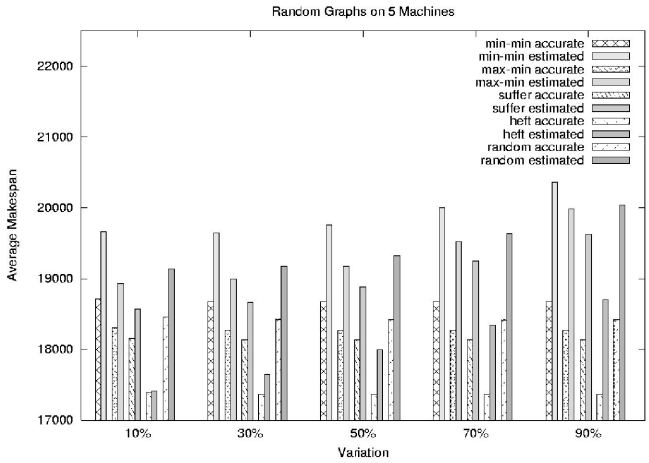


FIG. 4.3. *Effect of the variation of execution and communication times considering 5 machines.*

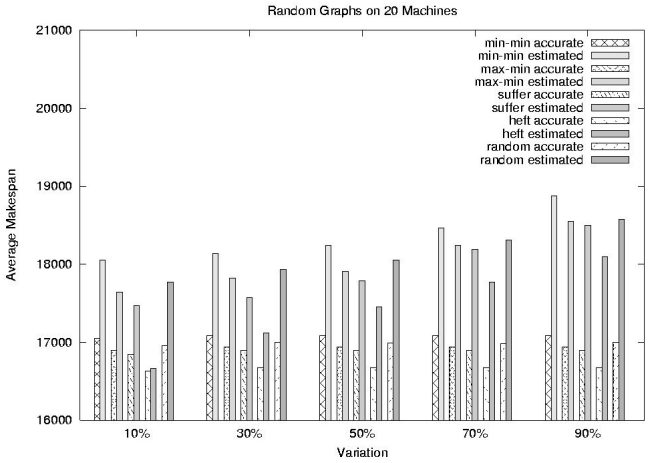


FIG. 4.4. *Effect of the variation of execution and communication times considering 20 machines.*

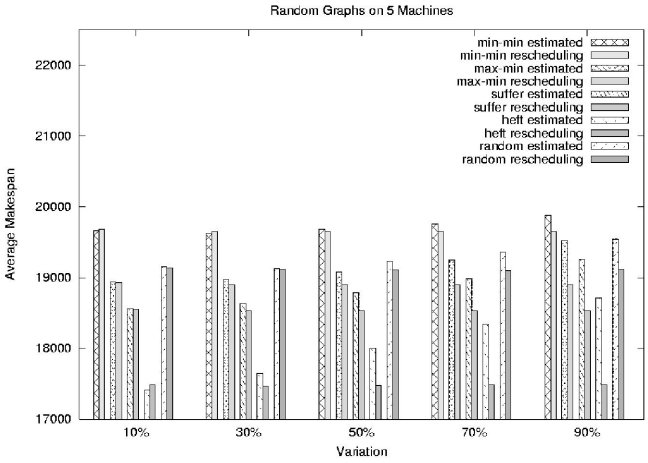


FIG. 4.5. *Rescheduling reduces makespan worsening. Case: 5 machines with execution time variations only.*

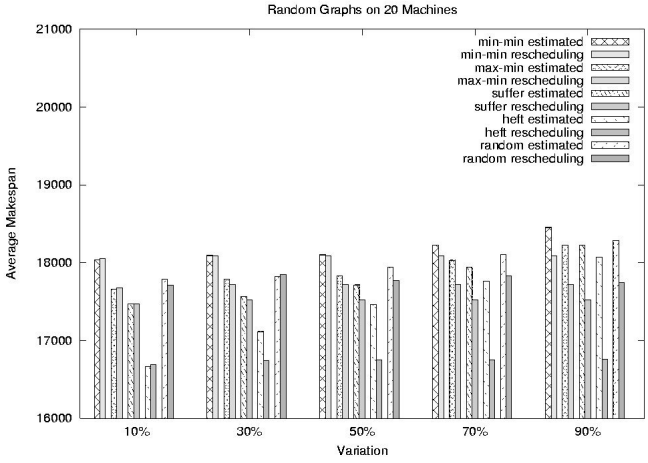


FIG. 4.6. *Rescheduling reduces makespan worsening. Case: 20 machines with execution time variations only.*

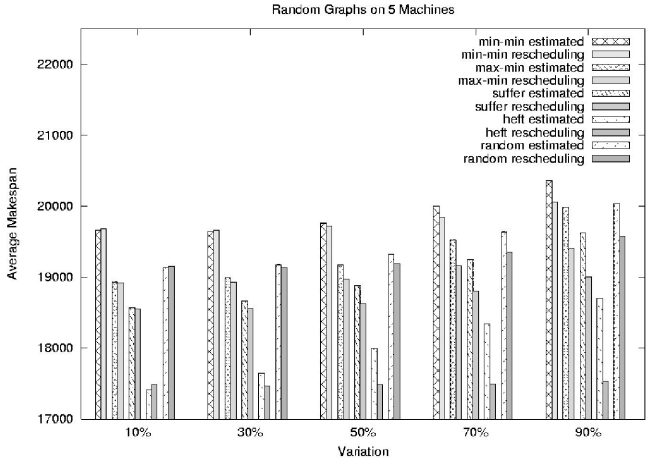


FIG. 4.7. *Rescheduling reduces makespan worsening. Case: 5 machines with execution and communication time variations.*

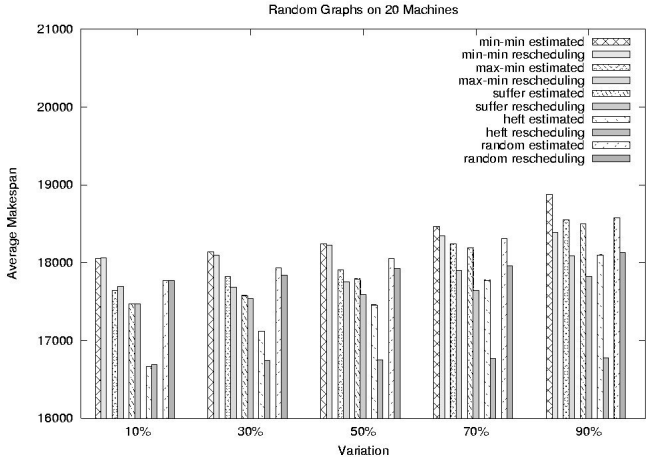


FIG. 4.8. *Rescheduling reduces makespan worsening. Case: 20 machines with execution and communication time variations.*

As it is shown in Fig. 4.9 and Fig. 4.10, the best results when rescheduling are achieved for the HEFT strategy. This is not surprising, since this is a list-based scheduling strategy that considers the DAG as a whole, while the remaining policies (i. e., non-list scheduling strategies) only consider those nodes ready to be executed. What is unexpected is the fact that min-min performs worse than the random policy. The performance of the remaining non-list scheduling policies (max-min and sufferage) is not entirely different from the performance achieved by the random policy.

4.4. Discussion. We now summarize the main results that have been derived from all the simulations performed.

Fig. 4.11 shows the average worsening percentage obtained for all the studied strategies, considering variations on the estimated execution time of 10% 30% 50% 70% and 90% when using 20 machines. For each strategy, the first number represents the worsening percentage when scheduling using estimated times, while the second number shows how this percentage is reduced when rescheduling.

As a consequence of the simulations carried out, we conclude that rescheduling performs well in terms of makespan in most cases, especially when task execution time is uncertain (variation of 70% and 90%). For example, the HEFT policy shows a worsening percentage of 8.4% when variation is 70%, and this effect is reduced to 1.2% when rescheduling. When the variation is low (10% and 30%), estimated execution time values are not far from real values, therefore the different policies do not frequently make erroneous decisions. In general, the higher the inaccuracy of estimated execution times, the greater the benefits of rescheduling.

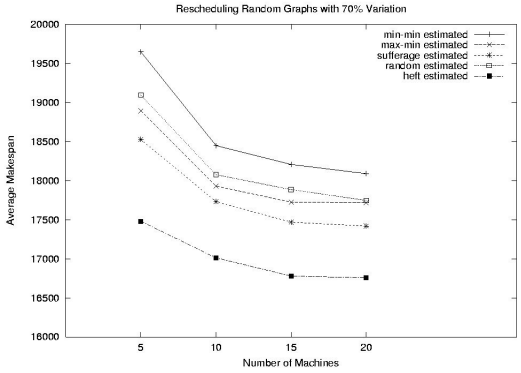


FIG. 4.9. *Overall strategy behavior when rescheduling in the presence of execution time variations only.*

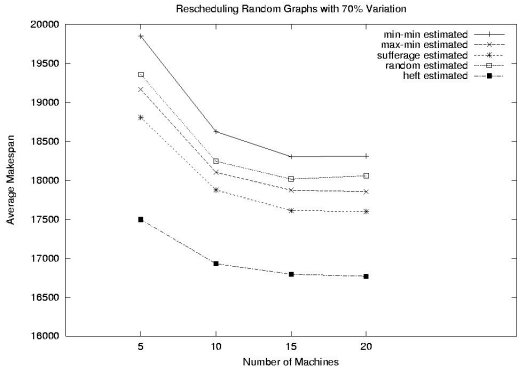


FIG. 4.10. *Overall strategy behavior when rescheduling in the presence of execution and communication time variations.*

In general, HEFT (list-based heuristic) significantly outperforms non-list based heuristics in both static and dynamic situations, under almost all the variations considered. Therefore HEFT constitutes a very suitable strategy for workflow scheduling on the grid.

		Variation				
		10%	30%	50%	70%	90%
Min-min	Estimated	5.5	5.5	5.7	6.1	7.5
	Resched.	5.6	5.6	5.5	5.6	5.5
Max-min	Estimated	2.7	2.9	4.0	5.9	9.0
	Resched.	3.0	2.9	3.0	3.1	3.2
Sufferage	Estimated	1.7	2.3	3.5	5.5	8.8
	Resched.	1.9	1.7	1.9	1.7	1.7
HEFT	Estimated	0	0.7	3.8	8.4	13.2
	Resched.	0.7	0.9	0.9	1.2	1.6

FIG. 4.11. *Worsening percentages.*

5. Conclusions. In this paper, we have analyzed the sensitivity to inaccurate execution and communication times of several heuristics used to schedule computational workflows on the grid. We compared the performance of a list-scheduling strategy (HEFT) and four non-list scheduling strategies (min-min, max-min, sufferage and random) both when applied as a pure static strategy or as a pure dynamic strategy computing a new schedule each time a new task is completed. Our results show that, in a system with heterogeneous

processors exhibiting congruent behavior, HEFT outperforms the other strategies in all cases, even where task computation and task communication times show a high variation at run-time. Interestingly, static schedules obtained by HEFT perform well for low-variation degrees (less than 50%). HEFT significantly benefits from a dynamic scheduling when time variations are greater than 50%. In practice, HEFT constitutes the most attractive strategy because the effectiveness of its dynamic version could be combined with a selective rescheduling policy, as presented in [14], in order to avoid high run-time overheads.

REFERENCES

[1] I. Foster, C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure*, San Francisco: Morgan Kauffman, 1999.
[2] *GriPhyN: The Grid Physics Network.*, `http://www.griphyn.org`
[3] *European DataGrid Project*, `http://www.eu-datagrid.org`
[4] *GridLab: A Grid Application Toolkit and Testbed*, `http://www.gridlab.org/`
[5] F. Berman et al, *The GrADS Project: Software Support for High-Level Grid Application Development*,International Journal of High Performance Computing Applications, Vol. 15, No. 4, 327–344 2001.
[6] *The CrossGrid Project*, `http://ww.crossgrid.org`
[7] *Ligo Scientific Collaboration*, `http://www.ligo.org/`
[8] S. Hastings, T. Kurc, S. Langella, U. Catalyurek, T. Pan, and J. Saltz, *Image Processing on the Grid: A Toolkit or Building Grid-enabled Image Processing Applications*, 3rd International Symposium on Cluster Computing and the Grid, 2003.
[9] *Atlas Collaboration*, `www.cern.ch/Atlas/`
[10] R. Wolski, N. Spring, J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Journal of Future Generation Computing Systems, Vol. 15, Num. 5–6, pp. 757–768, October, 1999.
[11] H. Topcuoglu, S. Hariri, and M. Y. Wu, *Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing,*, IEEE Trans. Parallel and Distributed Systems, Vol. 13, no. 3, pp. 260–274, 2002.
[12] A. Radulescu and A. J. C. van Gemund, *On the complexity of list scheduling algorithms for distributed memory systems*, Proc. ACM International Conference on Supercomputing, 1999.
[13] R. Sakellariou and H. Zhao, *A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems*, Proc. Of the 13th Heterogeneous Computing Workshop (HCW04), 2004.
[14] R. Sakellariou, H. Zhao, *A Low-Cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems*, Scientific Programming, 12 (4), pp. 253–262, Dec. 2004.
[15] Min-You Wu, Wei Shu, Hong Zhang, *Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems.*, 9th Heterogeneous Computing Workshop,2000.
[16] Y.-K. Kwok, I. Ahmad, *Static scheduling algorithms for allocating directed task graphs to multiprocessors*, CM Computing Surveys, 31(4), pp. 406–471, 1999.
[17] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, R. F. Freund, *Dynamic mapping of a class of independent tasks onto heterogeneous computing systems*, Journal of Parallel and Distributed Computing, 59(2), pp. 107–131, 1999.
[18] C. Raghavendra, A. Alhusaini, V. Prasanna, *A Framework for Mapping with Resource Co-Allocation in Heterogeneous Computing Systems*, 9th Heterogeneous Computing Wksp.: HCS 2000 (Cancun, Mexico, 1 May 2000), pp. 273–286.
[19] A. Mandal et al, *Scheduling Strategies for Mapping Application Workflows onto the Grid*, In 14th IEEE Symposium on High Performance Distributed Computing (HPDC 2005). IEEE Computer Society Press.
[20] R. Van der Wijngaart, M. Frumkin, *NAS Grid Benchmarks Version 1.0* , NASA Technical Report NAS-02-005 July 2002.
[21] 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, pp. 181, August 2001.

Edited by: Dana Petcu
Received: June 30, 2007
Accepted: July 26, 2007

Scalable Computing: Practice and Experience
Volume 8, Number 3, pp. 313–324. http://www.scpe.org
ISSN 1895-1767

A MOBILE AGENTS BASED INFRASTRUCTURE TO DELIVER VALUE ADDED SERVICES IN SOAS

ROCCO AVERSA*, BENIAMINO DI MARTINO* , SALVATORE VENTICINQUE* , AND NICOLA MAZZOCCA†

Abstract. The integration of Web Services and Software Agents brings new opportunities to deliver value added services in SOAs. This paper describes the functional model, the architecture design and the prototypal implementation of a platform for services adaptation and delivery. Agents technology is exploited to reconfigure the server platform to adapt the service provision to heterogeneous and handled devices. Virtual services are built on the platform for each client profile by conditioning the SOAP requests coming from WS requestors and adapting the returned responses. A general straightforward approach to extend client application in order to exploit platform facilities is described. Mobile agents are able to access and compose services accessing them by standard interfaces. User authentication, service discovery and publication, personalization are other provided facilities. Technological interoperability is provided by Web Services technology.

1. Introduction. The increase in size and performance of computer networks is both cause and effect of their own ubiquity and pervasiveness. This means that the user is able to exploit network connectivity without expensive add-on and wherever he is. Another relevant phenomenon is the increasing availability of handled mobile devices and their increasing capability through which the users are able to access Internet. In such a context a client could be able to discover countless resources without to be able to exploit most of them. In fact in order to grant the service usability it needs to care about functionalities, technological characteristics and limitations of client devices (mobile phone, palm, notebook). Further we have to consider the possibility that an user accesses the system at different times with various devices and he requires the service by a first connection mode and later collects the results by another one. A very interesting approach to design and develop a platform that provides access to distributed services by heterogeneous handled devices come from the adoption of "Mobile Agents". "Mobile Agents" have to be intended as both a technology and a programming paradigm. A mobile agent is a program able to migrate across the network together with its own code and execution state. We have experience in development of mobile agent platforms for Service Oriented Applications (SOA) [1, 2, 3, 4]. We describe here the design and the implementation of a reconfigurable Web Service architecture (WS for short) based on mobile agents technology. It has been developed upon the MAgDA toolset [5], a Mobile Agents based middleware for distributed and Grid programming. The Mobile Agent paradigm has been exploited both for services management and for service provision. In the first case agents implement the system infrastructure by providing efficient service discovery and composition, user authentication and authorization, session management and other middleware facilities. In the second case agents implements a service themselves which are able to be invoked by a Web Services Interface, further they can migrate form a host to another to optimize the system performance and utilization at server side, or to reconfigure the user device at client side. We present here the functional model and the component based architecture of the platform. The prototypal implementation of the platform was developed as a proof of concept of research activities concerning the CRDC project[1]. It was successively extended in collaboration with the Schumblerger-Sema as part of the activities of the Serviceware European project to investigate the design and development issues of a platform for providing Value Added Services to heterogeneous mobile devices. Hence this contribution has both a methodological and an experimental approach, aiming at the definition of new techniques and the application of new technological solutions to realize reconfigurable WS platforms based on the exploitation of mobile agents. Themes of interest we deal with are: mobile agent based programming paradigms for service provision and composition; service discovery and personalization; session management and system reconfiguration; design of open and interoperable systems; mobile computing and heterogeneous handled devices. In the second section we introduce the state of art of techniques and technologies involved in our activities. In the third and in the fourth sections we present the functional model and the architecture of the developed platform. In the fifth section the interaction protocol between client and middleware is described. Finally an overview of the functionalities provided by the available prototype is shown.

*Dipartimento di Ingegneria dell' Informazione, Second University of Naples, via Roma 29, Aversa, Italy {rocco.aversa, beniamino.dimartino}@unina2.it, salvatore.venticinque@unina2.it
†Dipartimento di Informatica e Sistemisitca, University Federico II of Naples, via Claudio 21, Napoli, Italy, n.mazzocca@unina.it
[1]Regional Competence Center on Information and Communication Technologies funded by Campania Reg. government

2. State of Art. QoS management is critical for distributed SOAs because of clients with different QoS requirements and also service providers with different resources, workloads and ever-changing business rules [6]. Concepts and guidelines from ISO/IEC QoS Framework [7] form the basis for the design and implementation of QoS management in SOAs. Recently, new ideas have been proposed to extend the functionality of UDDI registry by introducing additional features such as UX [8] and UDDIe [9]. Here, a QoS model (with QoS certifiers used to verify the claims of web service's QoS) is presented. [10] provides a generalized approach for building a QoS-based registry for SOA. This framework enhances the current UDDI standard to improve the efficiency and the quality of the discovery process, by taking in account user specified QoS parameters. Others propose a new infrastructure for QoS management such as WSLA [11] or SLAng [12] where the authors address the dynamic selection of services via an agent framework coupled with a QoS ontology. [13] proposes to exploit correlations between QoS properties of each Web service to provide good candidates of Web services for the matchmaking process when service brokering is performed. [14] proposes a quality of service assessment model that is able to capture the reason behind the ratings. This allows us to perform quality of service assessment in context, by using only the ratings that have similar expectations. [15] presents an integrated QoS management architecture and solution for the publish/subscribe style of enterprise SOA.

Our solution implements the QoS management architecture at the middleware layer as a service for Information Brokers. Major contributions include a general integrated QoS management architecture, a flexible and extensible XML-based QoS language, innovative resource models and exploitation of mobile agents technology.

A mobile agent is a program able to migrate across the network together with its own code and its execution state. This paradigm allows both a `pull` and a `push` execution model [16], in fact the user can choose to download an agent or to move it to another host. The agents are able to perform the computation where resources have been allocated in order to optimize the execution time by reducing the network traffic and the interactions with remote systems. Two forms of mobility characterize the migration of agents:

- `Strong mobility` is the ability of an Mobile Code System (called strong MCS) to allow migration of both the code and the full execution state of an agent (very close to process migration).
- `Weak mobility` is the ability of a MCS (called weak MCS) to allow code transfer across different Computing Environments (CEs) together with the intermediate values of those data which are relevant to resume the status of the application: the activation stack and complete image of the process could not be migrated, the application is resumed, not the process.

We will intend the second one thorough the paper. Web services [17] are an emerging distributed computing paradigm which focus on Internet-based standards technology such as XML and HTTP. SOAP, WSDL and Ws-Inspection. They have been designed to support interoperability, i. e. independence of the transport protocols, programming language, programming model and system software. For our purpose the web services support the dynamic registering and discovery of services in heterogeneous environments. WS-Inspection [18] defines a simple XML language and related conventions for locating service descriptions published by a server provider. The service is usually a URL to a WSDL document. WSDL [19] is an XML document for describing Web services. It can be referenced in a Universal Description, Discovery and Integration (UDDI) register [20]. Integrating Web services and software agents brings new opportunities and helps in defining new services. Once this integration is realized, software agent concepts and technologies will help to enable new and advanced operational and usage modalities of Web services. Several publications have been published to support this thesis, including [21, 22]. The Web Services Architecture Specification of the W3C [23] foresees itself the integration of agent technology in Service Oriented Architectures, "... software agents are the running programs that drive Web services - both to implement them and to access them as computational resources that act on behalf of a person or organization". Some interesting results can be found in Lyell et al. [22], which discuss the concept of a hybrid J2EE/FIPA-compliant agent system, in [24, 25], where the authors have proposed adoption of agents for service integration, or in [26, 27] which uses mobile agents and web services for management of virtual home environments. Especially agent migration techniques, allow to dynamically find the "best" node where to execute the service at server side [25, 28, 29]; to upload/download the embedded client application to reconfigure the user device [30].

3. The functional model. The model defined in the CRDC project is shown in figure 3.1. It is composed of the three blocks listed here:

1. A client device is characterized by its technology, the amount of memory, the computing power, the supported communication protocols, the connection bandwidth;

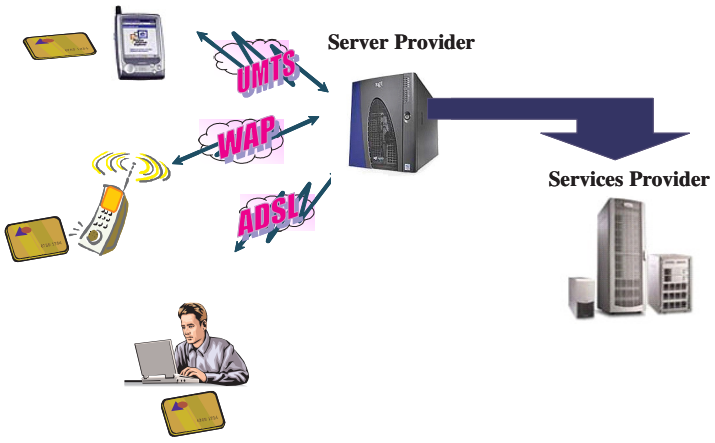


FIG. 3.1. *The CRDC model*

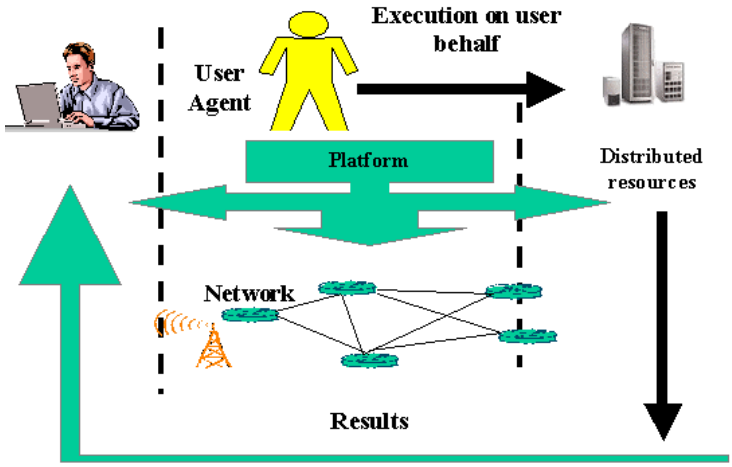


FIG. 3.2. *The Agent based approach*

2. A Services Provider offers the end-user services providing different implementations for different kind of client profiles (user, device, session profile);
3. A Server Provider represents the platform that supports the interaction between the client and the services provider, overcoming the challenges which could arise in service exploitation.

The main challenge in the exploitation of web services through mobile devices is the overcoming of their hardware and software limitations. Dynamic discovery, late binding and services composition cannot be supported by a wide class of smart devices. It motivate developers to experiment new methodology of exploiting remote resources to extend the capability of the client devices. Here we use the Mobile Code technology for local device reconfiguration and for exploitation of the back-end nodes in order to overcome the client limitations. When the execution of the agent is supported by the client device agent migration is exploited to reconfigure the terminal and to allow the user to play the service. When the device does not support the full execution of the client application some activities must be performed by remote resources. The agent executes on the server and provides the access to the service by simpler interfaces. As it is shown in Figure 3.2 the user interacts with a personal agent that is able to migrate on the server and to perform the required actions on behalf of its owner. The proactive characterization of the agent allows the autonomous completion of a task according to a delegation based model that provides the agent with the authorization profile of this owner. In fact in order to represent a human user, the delegate must be able to characterize the context of the interaction. The user context can be composed of actual temporal and spatial location, terminal device profile (configuration and

technology), user's preferences, nearby users, nearby resources, noise level, network connectivity, communication cost, communication bandwidth, and social situation. Some of this contextual information can be sensed through sensors like GPS, and network bandwidth detection, while others can be inferred from knowledge bases. Each of these contextual parameters is fairly complex. For example, terminal device characteristics could involve audio capability, video capability, image decoder, text capability, local storage capacity, screen size, network capability and content mark-up language capability. In a feasible example of use of the system:

1. The customer accesses the platform by its device, hence the system identifies the user and characterizes the access mode.
2. A session profile is built according to the device features and the connection parameters.
3. A personal agent is built and is initialized with both the user profile and the session profile by which the security level, the best appreciable QoS are defined.
4. The agent shows to the user the set of functionalities he is allowed to.
5. The user is able to ask for a service. An available version is discovered and brokered according to the to the service requirements, the session profile and the user preferences.
6. The client side of the application is provided by the service vendor; it is retrieved by the agent and forwarded to the user as a web page, a mobile agent, an applet or an embedded application that needs to be downloaded and installed.

According to a first coarse grain classification [31], we suppose here to have four classes of client devices. The considered devices types are different one from another because of their memory capacity, their computational resources, their display characteristics, their allowed connection mode and so on. Depending on the above features the platform and the client should be able to broker the best set of parameters which allow the session to be opened. Some of these parameters are the communication protocol, the Human Computer Interface (HCI), what applications have to be executed locally or remotely, the level of trustful, etc

1. A first class of devices includes all the laptops, pc and more powerful machines, which have no limited resources in our application fields.
2. The second one is composed of those devices, like palms, which have its own embedded Operative System and are able to download and install some software in order to perform any kind of application, but have limited resources. Here the user could choose to reserve the minimum amount of resources but waiving a good level of interactivity.
3. JAVA enabled smart phones belong to this class. They are able to access the system by downloading an APPLET that implements the application client.
4. The last class is composed of that handheld devices which do not support reconfiguration. They allow just the browsing of a web pages, implemented in the HTML, WML or similar languages.

Many solutions can be conceived by different vendors for developing client applications targeted to different classes of devices.

4. System architecture. The system architecture has been designed aiming at two main targets. The first one is the interoperability, in order to support the integration of new third party services. The second one is the automatic management and composition of services by software agents able to access well known standard interfaces. In Figure 4.1 is shown a component-based model of the designed architecture. Some components provide basic facilities and are embedded in the platform, some other functionalities can be integrated as external services accessible by a Web Services interface. In the first prototypal version the Mobile Agent engine is implemented by the ASDK (Aglet Software Development Kit), but we have already developed a porting to JADE, a better supported and FIPA standard agent platform. A mobile agents platform supports the mobile agent development and execution providing an Agent server and the basic mechanisms of the programming paradigm such as agent creation, cloning, migration and communication. The original platform was extended in order to integrate new features at different levels. Some facilities have been integrated inside the agent server, the other are external components or they have been developed as application agents. The platform behaves both as a services provider and as a services requestor. In the first case Web Service are exploited to provide vendors with the possibility:

- to deploy their services;
- to deliver them with added values by services composition;
- to exploit the Virtual Home Environment of the platform customers (authorization, personalization, ...).

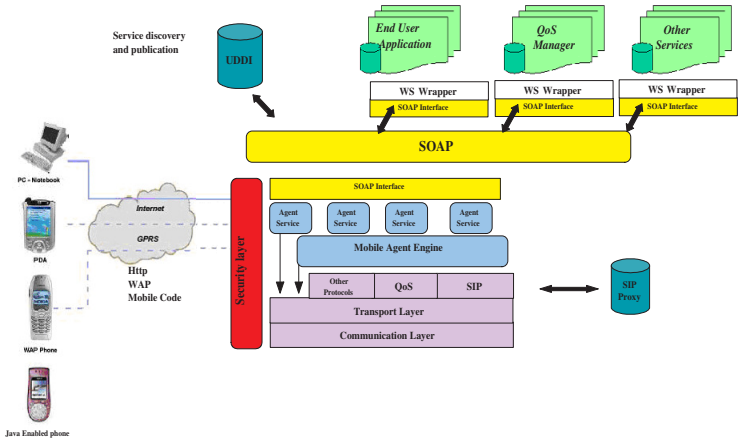


FIG. 4.1. *The system architecture*

Furthermore it allows service requestors:

1. to broker the best implementation according their client profile;
2. to exploit transparently through the same interface different service which would not be usable without the support of the platform.

In fact the platform is able to exploit the implementations deployed by trusted providers and composes them to build new value added services or to increase the availability of the existing ones building multiple alternatives. It supports the user in their exploitation and personalizes the access in order to optimize the customer satisfaction. The UDDI register is employed to support the publication of new services by the vendors and to discover the available implementations. Searching for a service implementation the client gets the WSDL description that provides methods and parameters of the implemented interface. Web services supports a kind of "technological" interoperability. The WSDL interface and SOAP protocol allows the service invocation by any kind of application, rather than by a human user. Nevertheless another issue to be addressed is related to the automatic exploitation of the services. The question is how the requestor should invoke the methods of the provided interface to exploit the service ? What is the behavior of the selected implementation ? What are the input/output function implemented by each method ? In order to overcome these problems many solutions have been proposed in related works, but we are still dealing with an open issue. Some of them deal with the automatic exploitation of the discovered services basing on the definition of standard taxonomies. Some others investigate how to understand the way of using an application exploiting a semantic analysis of a formal or semi-formal description of the service functionalities. To support the services integration and their exploitation we conceived the following model that allows the plug-in of new components both to expand the platform functionalities and to deliver new kind of services to the customers. The plug-in model is shown in Figure 4.2. We can see an agent as a new functionality provided by the platform (we can also have more agents collaborating to provide just a single functionality). Every agent is able to drive a class of services by a standard interface that needs to be implemented by the providers in order to deliver their service through the platform. In our model a service is characterized by a 3-tuple of elements:

1. a class,
2. a wsdl interface,
3. an agent driver.

The class is identified by a label and represents all the services which can be exploited by the same interface. In order to deploy a service within a chosen class the provider needs to implement the defined interface. Like the Operative System is able to drive a peripheral device through a program driver, in the same way the platform exploits the generic implementation of the discovered service within a class by a software agent. In order to handle a new kind of service the platform administrator needs to define a class (label, interface) and to implement the agent driver for that service. When the provider is publishing a new service he needs to select the class and to provide its WSDL together with some other information. A software authority performs a check to verify the compliance between the interface described in the WSDL and the one required in the selected class.

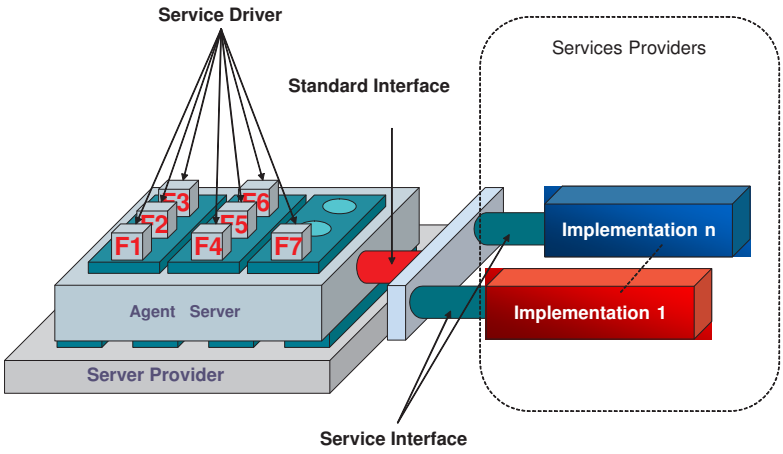


FIG. 4.2. *The service integration*

If the check is successful the new service implementation is published in the UDDI register. Note that this deal with a syntactic compliance but we do not grant a semantic correctness yet. Another kind of agreement could be checked by defining a set of test to be performed before to make available the service (we could check the output when a set of defined data is provided in input). Some other functionalities as the access manager and the agent localization facility are embedded in the platform; we mean they have been developed by extending the agent platform or providing new APIs or applications. We will deal with them in the following making clear the semantic of some components depicted in Figure 4.1 as the SIP register and the security layer. Some examples are:

- device and server localization;
- remote client management (upgrade, healing, backup, reconfiguration, ...);
- personalization (profiles management , context awareness, HCI , content management);
- services management (service integration, deployment and delivery, service reconfiguration);
- billing;
- user information (address book, remainders, e-mail)

Actually a lot of prototypal components and protocols have been developed. Some stand-alone applications are available to show how the new functionalities work, further a prototypal framework that integrates some of them is provided. The prototypal implementation is distributed on three workstations with the following roles: a server provider, a register, a service provider. The server provider supports the authentication of the user by different mechanisms, service publication, removal and discovery. Further it supports the hosting of the user agents (when the the device is not suited to execute them) and grants the persistence and the resumption of the active services. All the interaction with the register, such as services discovery and publication, are performed at destination exploiting agent migration. An agent is dispatched where the register resides in order to perform the search locally and to return just the results (we minimize communication and we support proactive connectionless discovery). In our prototypal implementation the services are implemented by mobile agents themselves and they are accessible by a Web Services interface. The agent server has been extended in order to be executed as a web application. Its Web Services interface exports those methods which allow to interact with agent based application and to exploit basic agent mechanism.

5. Interaction model. A relevant issue is how the client application can exploit the platform facilities and the services delivered. Here we propose a common approach that can be applied in general and does not depend by a specific technology. The interaction model among client, middleware and service provider allows to design and develop each software module independently. The middleware developer looks at services and clients as black box entities. Transparent access to the services must be provided. When they exists, client

```
public interface MagdaWS extends Remote {
public String connect(String login, String pwd, String profile);
public void logout(String token);
public String[] listServiceType(String token);
public String find(String serviceType, String token);
public String getResponse(String requestID);
public String[] getTokenList(String token);
public void closeSession(String token);
}
```

FIG. 5.1. *MAgDA Web Service interface*

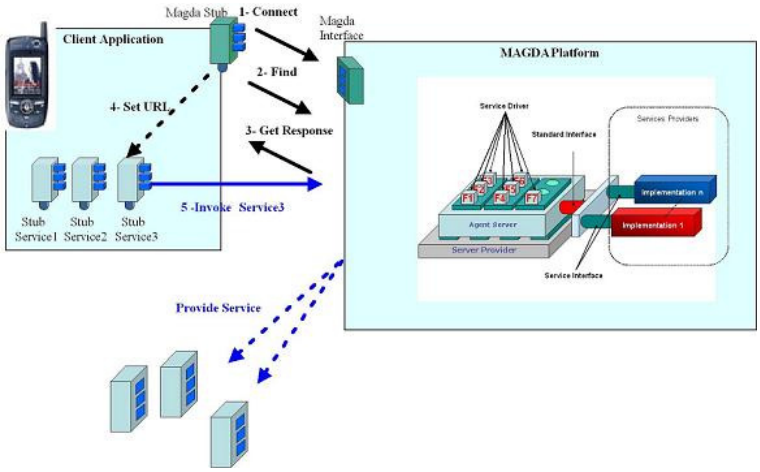


FIG. 5.2. *The middleware interaction protocol*

self-reconfiguration capabilities are unknown by the middleware. As any other requestor the client application will be conceived to work with a well known kind of services whose wsdl has been defined and has been made publicly available by the provider. The application will invoke remote services by a set of stubs (S1, ..., Sn), which have been built by their wsdl. The extension of the client application in order to exploit the middleware facilities is straightforward. The interactions among requestors and the platform take place just before to invoke a service and they initialize the session before the application is starting. At client side, during the configuration phase, the information necessary to bind the stubs to the right providers is collected. Meanwhile, at middleware side, the required services are brokered and the platform is reconfigured to support the service exploitation by the new client profile. Platform reconfiguration is exploited to adapt the service exploitation but also to allocate the necessary resources for an effective dispatching of the incoming workload.

As it is shown in Figure 5.2 the new client will be composed of the original application and of an access manager (in Figure MS—Magda Stub) that is used to exploit the facilities provided at middleware-side.

The MAgDA WS interface is shown in Figure 5.1.

In Figure 5.2 the interaction protocol between the client and the middleware is described.

The *connect* method allows the client to access. It needs to communicate the user credentials and its profile for that session. For each successful authentication a personal agent is created in any container of the platform which can be distributed geographically. The token returned identifies the personal agent for that session and must be used for all the following requests. Let us consider that Web Service technology allow to deploy stateless services. Here personal agents represent active sessions for logged users. The *logout* method closes the session without disposing the personal agent. The *closeSession* method disposes the personal agent. In order to know those services which can be exploited by the platform the *listServiceType* method can be invoked. The *listServiceType* method returns the list of service categories which are available in the platform and the wsdl address for each of them. If the exploitation of the desired service is supported by the platform the client can ask for the binding information needful to initialize its stubs. The wsdl is used to build the service interface while the binding must be retrieved whenever a new session is opened because it depends by the client

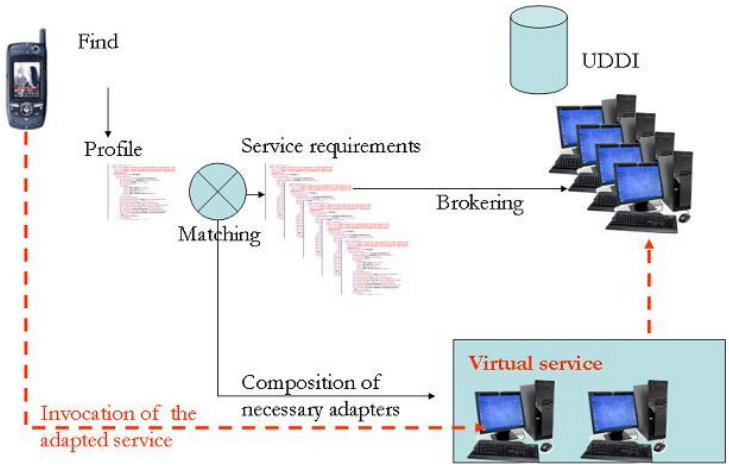


FIG. 5.3. *The middleware interaction protocol*

profile and by the dynamic condition of the system. The *find* method is used to broker the service and returns a *requestID* that identifies the request. Its invocation is not blocking. The received token will be used to get the response, when it will be available, by the *getResponse* method. The obtained link will refer to the brokered service. The result will be used to initialize the client stub before to start the application. The *getTokenList* method retrieves the list of requestIDs of all pending requests belonging to the session identified by the same token.

The invocation of *find* method of MAgDA interface starts the procedure shown in Figure 5.3. For each available service able to provide the desired functionality a matching between the service requirements and the client profile is performed. If a compliant service is found, its address is returned to the client in order to bind the stub to the right provider. When the compliance cannot be granted by the service the platform selects the service implementation that fits as well as possible the client capabilities. Then it can provide the necessary adaptation exploiting internal components or composing external services. In this case the binding information returned to the client will refer to a virtual service that is executing on the platform. Here a set of agents, acting as a bridge, processes and throws client request from an agent to the other until the message reaches the brokered provider. In the same way the response flows back to the client across a set of adapters. A tunneling of the SOAP messages allow us to distribute the workload and to adapt the contents transparently. Message elaboration along the outward path is exploited to distribute the workload among different providers or to adapt the input parameters of the brokered services. Many instance of the same bridge can be distributed on different nodes in order to offer alternative paths, which can be exploited for different profiles, or which can be replicated to distribute the workload. The backward path is exploited to adapt the contents returned by the provider.

Other approaches have been conceived for different kind of devices to exploit at the best particular functionalities. Agents could be pushed on the client to perform a reconfiguration or a software update when their execution is supported locally. Of course this model needs a larger amount of computational resources and memory. Furthermore application developers must deal with mobile agents technology. When the user owns a smart-phone or a too simple handheld device, a personal agent that supports the interaction between the client and the services could be created on the remote machine. The agent accepts the user's requests and invokes on the server the desired services. The user interface and the replies are forwarded to the client by the agent. Also a specific implementation of the Human Computer Interface needs to be provided for the target client device. This different approaches are shown in Figure 5.4. Hybrid solutions can be conceived splitting the application in a client agent and a back-end agent when the limited resources cannot carry out the execution. Hence the platform provides to the user services and the resources to support their execution.

6. A prototypal implementation. The prototypal implementation of the described platform represents for us a test-bed for the integration of the experimented technologies and the test of the defined applications and protocols. As it is shown in Figure 6.1 the platform implementation is built upon a mobile agent server. Mobile agents implement the platform facilities; multiple instances can serve many clients at the same time. In order to support interoperability with different technologies, which are not agents based, we deployed the

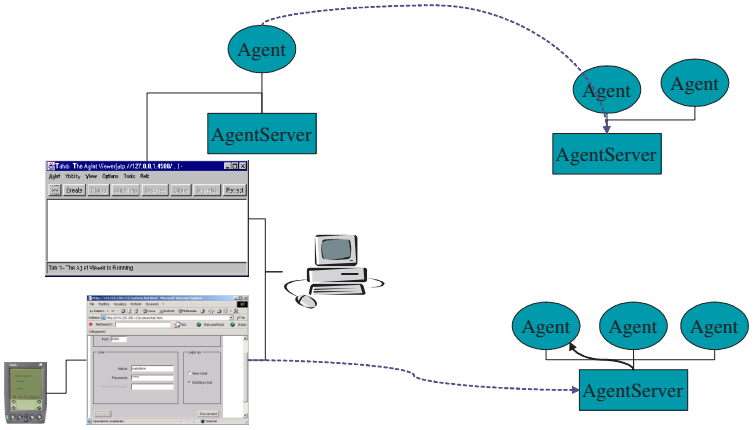


FIG. 5.4. *Interaction models*

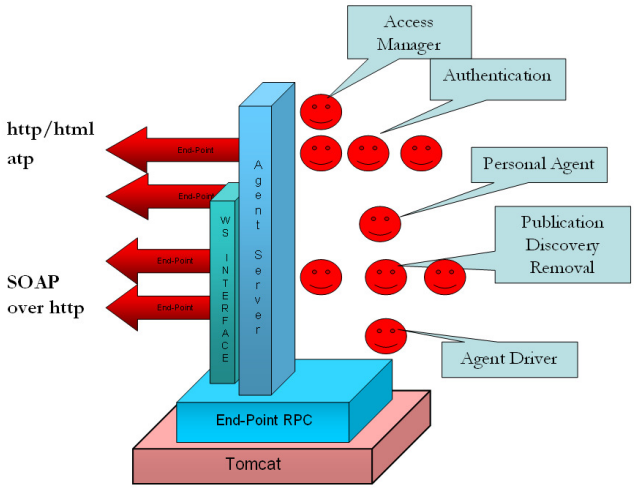


FIG. 6.1. *The Server Provider*

agent server as a web application inside the Tomcat Application server. In this way any requestor is able to invoke the platform facilities also by a web service interface. The application server needed to be patched in order to handle also the ATP protocol for agent communication and migration if the Aglet version is used, on the other hand the Jade version exploit the already supported http protocol. We provided a web interface for end-user services and an agent client for agent-enabled devices. When it needs the user is able to communicate, by a web browser, with the agents which are executing on the remote server. An agent can be identified by an URL and can handle a number of URL aliases. The http request is parsed and translated by the agent server using an agent communication language messages and it is forwarded to the agent in charge to handle it. The server provider is equipped with an agent server that hosts four resident running agents: an Access Manager and three authentication agents (each of them implements a different authentication mechanism). We provided weak (based on login and password) and strong authentication mechanisms (based on cipher algorithms). The simplest devices will support just the weakest ones and will allow the user to exploit the less security critical services. The authentication options offered by the Access Manager, at the state of art, are: 1) login and password, 2) challenge response based on the exploitation of a hash function, 3) challenge response with digital signature using a smart-card or 4) not. If the authentication is successful the user is able to be connected with his personal agents that could be already running or could need to be created. Each user is driven by his personal agent to exploit the platform facilities. In Figure 6.2 the administration panel is shown. A GUI

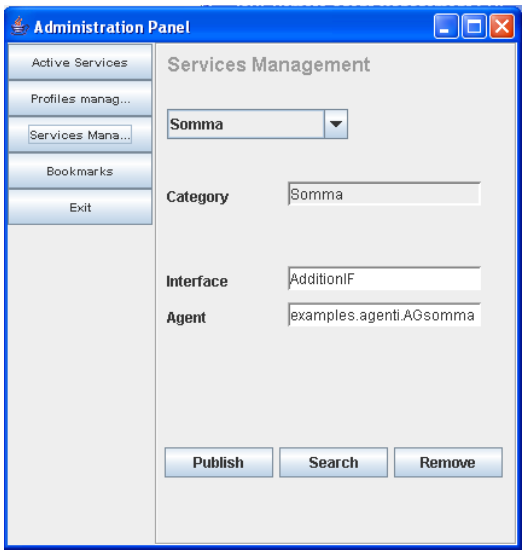


FIG. 6.2. *The administration panel*

panel implemented by a java agent can migrate on the user device and allows its owner to support asynchronous interactions and to reduce the on-line connection time. Therefore a web interface allows to exploit the services by a wide class of devices. The menu offers some already available facilities: active sessions, profiles management, services management and bookmarks. The first item allows the user to list his active requests (open sessions) and to resume or dispose the agents in charge of delivering the services. It needs when a user chooses to close a session and to resume it accessing the system by another device or from a different location. The second item allows the user to modify and store different profiles and to select the one he is exploiting in that moment. A profile is composed of personal information, device description (we are developing a facility to characterize automatically the user device), session properties, security level. A selected profile is exploited by the platform to personalize the discovery or to adapt the service. When it is supported by the devices an agent migrates to the client in order to collect automatically the profile information. Service management is described in detail in the following. It supports service publication, removal and discovery. The last item allows the user to list the saved bookmarks. The administration panel support the publication and the removal of new services, which are implemented by three mobile agents. The publication of new services can be exploited by a form that must filled with the WSDL address of the service and the class which identify that kind of application. An authority will automatically check the compliance of the services with the ones belonging to the same class by matching the WSDL with a pre-defined interface and will allow the publication. The discovery of a service is performed by a specialized agent who is able to migrate on the UDDI server and to perform the research close the data according to specialized algorithms. This approach optimizes the communications and the performance of the system. The results of discovery are filtered by a matching algorithm that compare the client profile with the service description. To select the suited implementations of a services among the published ones we perform a matching between two XML representations of user and service profile. We need that the service description and client profile should have the same XML schema and for each attribute should specify the minimum value required for the service exploitation. The matching algorithm is able to compute a distance between the two representation and to warn about the differences if it is required. The matching is performed by four steps:

1. Parsing e Hashing : each profile is represented as a DOM (Document Object Model Specification) tree (T1 and T2) and for each node an Hash value is computed.
2. Checking e Filtering : T1 and T2 are equivalent if the Hash value of the root nodes are equals. If T1 and T2 are different the algorithm looks for the differences comparing the sub-tree and branching the ones which result equals.
3. Matching: the algorithm computes the minimum-cost matching between T1 and T2 Mmin(T1,T2). Mmin(T1,T2) is a script that allows to transform T1 to T2 by a sequence made of the minimum number of simple edit operations (delete, insert,update) applied to the nodes.

4. Minimum-cost edit script generation: this provides the differences between the two XML documents.

Once the service has been chosen the next step is its invocation. We chose as case study a service for instant messaging. We defined the class "InstantMessaging" that is associated with a IMAgent and a IMInterface. An implementation of the services can be published into the UDDI register of our platform if its WSDL is compliant with the IMInterface. The definition of IMInterface methods and parameters is stored in an internal database that will be queried to check the compliance of a new implementation. To set up the demo we need to deploy the service and publish it by the platform administration panel. Any user now is able to discover the service and to enjoy it. When an implementation of a service belonging to the InstantMessaging class is invoked the platform creates an instance of the IMAgent and passes to it the URL of the service provider. The IMAgent books the service using the URL and receives some parameters for exploiting the service facilities (protocol, server address, session key,...). The IMAgent stores the session data, creates and sends an instance of the client application to the user device. If the user device is not able to host and execute the agent client a proxy agent is created. It provides a web interface and supports from remote the exploitation of the service. For example the proxy agent is able to handle the incoming call, , to store the list od contacts, the save the discussion, to set the status of the client (busy, not visible, on-line, ...), ... Of course the user can choose to disconnect his device and to reconnect by another one or from another location. In this case it will need just to authenticate himself again, to check the active services and to ask the right agent for a new player that will allow him to exploit the service resuming the state of the application. Finally we need to observe that these kind of services support the client and agent mobility. Both the devices and the agents can migrate across the network. The platform support the localization of its agents by a SIP (Session Initiation Protocol) that is used to resume the connection when the communication with agents fails. A common namespace managed by a SIP register allows to localize the agent and the hosting agent server.

7. Conclusions. In this paper we aimed to provide an overview of MAgDA, a software platform that provides to the users value added services by the composition and delivery of Web Services. Web Service Technology provides interoperability and the Mobile Agent programming paradigm is exploited to build the platform infrastructure. We described the functionalities of our system and its architectural model. We provided a WS interface to allow any WS requestor to exploit in a straightforward way advanced services provided by the platform. Finally we presented the state of art of a prototypal implementation and some mechanisms which have been integrated in the platform. Future works will deal with the integration of the all available components and the development of new functionalities. We designed and are already developing some services for automatic composition of web services and its choreographic execution. Furthermore we are going to terminate a porting to the Jade agent platform because better supported and compliant to the actual standards. Tests and measures in real application environments need to be planned and performed in order to evaluate availability, performance and scalability.

REFERENCES

[1] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Mobile agents for distributed and dynamically balanced optimization applications,* Proc. of the 9th International Conference on High-Performance Computing and Networking (HPCN Europe 2001), London, UK, Springer-Verlag (2001) 161–172.

[2] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Mobile agent programming for clusters with parallel skeletons,* Proc. of 5th International Conference on High Performance Computing in Computational Sciencesm, VECPAR'2002, Porto, Portugal (2002).

[3] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *A resource discovery service for a mobile agents based grid infrastructure,* Concurrent Engineering—Enhanced Interoperable Systems (2003) 1011–1017.

[4] Aversa R., Martino B. D., Mazzocca N., Venticinque S.: *Terminal-aware grid resource and service discovery and access based on mobile agents technology,* Proc. of 12th Int. Conference on Parallel and Distributed Processing (PDP 2004), A Coruna, Spain, IEEE Computer Society (2004) 40–45.

[5] R. Aversa, B. Di Martino, N. Mazzocca, S. Venticinque, MAGDA: *A Mobile Agent based Grid Architecture, Journal of Grid Computing, Springer Netherlands,* ISSN 1570-7873 (Print) 1572-9814 (Online), pp. 395–412, December 2006, Vol. 4 Num. 4.

[6] C. Wang, G. Wang, A. Chen, and al: *A policy-based approach for qos specification and enforcement in distributed service-oriented architecture,* In Proceedings of the 2005 IEEE International Conference on Services Computing IEEE Press, 2005.

[7] *I. O. for Standardization,* Iso/iec. international standard 13236 technology—quality of service: Framework, Dec. 1998.

[8] Z. Chen, C. Liang-Tien, B. Silverajan, and L. Bu-Sung: *Ux an architecture providing qosaware and federated support for uddi,* In Proceedings of the 2003 International Conference on Web Services, 2003.

[9] A. S. Ali, O. F. Rana, R. Al-Ali and D. W. Walker: *Uddie: An extended registry for web services,* In Proceedings of the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference. IEEE Press, 2003.
[10] A. Kumar, A. El-Geniedy, and S. Agarwal: *A generalized framework for providing qos based registry in service oriented architecture,* Proceedings of the 2005 IEEE International Conference on Services Computing. IEEE Press, 2005.
[11] A. Keller and H. Ludwig: *The wsla framework: Specifying and monitoring of service level agreements for web services,* Research report RC22456, IBM, 2002.
[12] D. D. Lamanna, J. Skene, and W. Emmerich: *Slang: A language for defining service level agreements,* Proceedings of the 9th IEEE Workshop on Future Trends of Distributed Computing Systems. IEEE Press, 2003.
[13] L. Taher, R. Basha, and H. E. Khatib: *Establishing association between qos properties in service oriented architecture* Proceedings of the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference. IEEE Press, 2003.
[14] V. Deora, J. Shao, W. A. Gray, and N. J. Fiddian: *Supporting qos based selection in service oriented architecture,* Proceedings of the International Conference on Next Generation Web Services Practices. IEEE Press, 2006.
[15] G. Wang, A.Chen, C. Wang, C. Fung, and S. Uczekaj, *Integrated quality of service (qos) management in service-oriented enterprise architectures,* Proceedings of the 8th IEEE Intl Enterprise Distributed Object Computing Conf. IEEE Press, 2004.
[16] Xu C. Z., Wims B.: *A mobile agent based push methodology for global parallel computing,* Concurrency—Practice and Experience **12** (2000) 705–726.
[17] Kreger H.: *Web services conceptual architecture(wsca 1.0)* (2001).
[18] Brittenham P.: *P. an overview of the web services inspectin language* (2002).
[19] R. Chinnici, M. Gudgin, J. M. S. W.: *Web services description language (wsdl) version 1.2* (2003).
[20] OASIS: *Uddi technical white paper,* Technical report, `http://www.uddi.org` (2000).
[21] Maximilien E., Singh M.: *Agent-based architecture for autonomic web service selection,* Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia (2003).
[22] Lyell M., Rosen L., Casagni-Simkins M., Norris D.: *On software agents and web services: Usage and design concepts and issues,* Proc. of the 1st International Workshop on Web Services and Agent Based Engineering, Sydney, Australia (2003).
[23] cGroup W. W. S. A. W.: *Web service architecture recommendation (2004)* `http://www.w3.org/TR/2004/NOTE-ws-arch-20040211`
[24] Maamar Z., Sheng Q. Z., Benatallah B.: *Interleaving web services composition and execution using software agents and delegation,* Proc. of Workshop on Web Services and Agent-based Engineering (AAMAS2003). (2003).
[25] Maamar Z., Sheng Q. Z., Benatallah B.: *On composite web services provisioning in an environment of fixed and mobile computing resources,* Information Technology and Management **5** (2004).
[26] Ragusa C., Liotta A., Pavlou G.: *Dynamic resource management for mobile services,* Proc. of 5th International Workshop on Mobile Agents for Telecommunication Applications (MATA-03), (2003).
[27] Bohoris C., Pavlou G., Liotta A.: *Mobile agent-based performance management for the virtual home environment,* Journal of Network and Systems Management **11** (2003).
[28] Kifer M., Lara R., Polleres A., Zhao C., Keller U., Lausen H., Fensel D.: *A logical framework for web service discovery,* Proc. of Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications (ISWC 2004). Volume 119, Hiroshima, Japan (2004).
[29] Buhler P. A., M. Vidal J.: *Semantic web services as agent behaviors,* Burg B. et al. (eds), Agentcities: Challenges in Open Agent Environments, Springer-Verlag (2003) 25–31 `http://jmvidal.cse.sc.edu/papers/buhler03a.pdf`
[30] L. Bettini, R. De Nicola M. L.: *Software update via mobile agent based programming,* Proc. of SAC 2002, Madrid, Spain, ACM 1-58113-445 (2002).
[31] *3rd Generation Partnership Project Technical Specification Group Services, Aspects, S.:* Mobile station application execution environment (mexe); functional description; stage 2; arib std-t63-23.057, Technical report, 3GPP (2001).

Edited by: Dana Petcu
Received: July 15, 2007
Accepted: August 6, 2007

AIMS AND SCOPE

The area of scalable computing has matured and reached a point where new issues and trends require a professional forum. SCPE will provide this avenue by publishing original refereed papers that address the present as well as the future of parallel and distributed computing. The journal will focus on algorithm development, implementation and execution on real-world parallel architectures, and application of parallel and distributed computing to the solution of real-life problems. Of particular interest are:

Expressiveness:

- high level languages,
- object oriented techniques,
- compiler technology for parallel computing,
- implementation techniques and their efficiency.

System engineering:

- programming environments,
- debugging tools,
- software libraries.

Performance:

- performance measurement: metrics, evaluation, visualization,
- performance improvement: resource allocation and scheduling, I/O, network throughput.

Applications:

- database,
- control systems,
- embedded systems,
- fault tolerance,
- industrial and business,
- real-time,
- scientific computing,
- visualization.

Future:

- limitations of current approaches,
- engineering trends and their consequences,
- novel parallel architectures.

Taking into account the extremely rapid pace of changes in the field SCPE is committed to fast turnaround of papers and a short publication time of accepted papers.

INSTRUCTIONS FOR CONTRIBUTORS

Proposals of Special Issues should be submitted to the editor-in-chief.

The language of the journal is English. SCPE publishes three categories of papers: overview papers, research papers and short communications. Electronic submissions are preferred. Overview papers and short communications should be submitted to the editor-in-chief. Research papers should be submitted to the editor whose research interests match the subject of the paper most closely. The list of editors' research interests can be found at the journal WWW site (`http://www.scpe.org`). Each paper appropriate to the journal will be refereed by a minimum of two referees.

There is no a priori limit on the length of overview papers. Research papers should be limited to approximately 20 pages, while short communications should not exceed 5 pages. A 50–100 word abstract should be included.

Upon acceptance the authors will be asked to transfer copyright of the article to the publisher. The authors will be required to prepare the text in LaTeX 2ε using the journal document class file (based on the SIAM's `siamltex.clo` document class, available at the journal WWW site). Figures must be prepared in encapsulated PostScript and appropriately incorporated into the text. The bibliography should be formatted using the SIAM convention. Detailed instructions for the Authors are available on the SCPE WWW site at `http://www.scpe.org`.

Contributions are accepted for review on the understanding that the same work has not been published and that it is not being considered for publication elsewhere. Technical reports can be submitted. Substantially revised versions of papers published in not easily accessible conference proceedings can also be submitted. The editor-in-chief should be notified at the time of submission and the author is responsible for obtaining the necessary copyright releases for all copyrighted material.