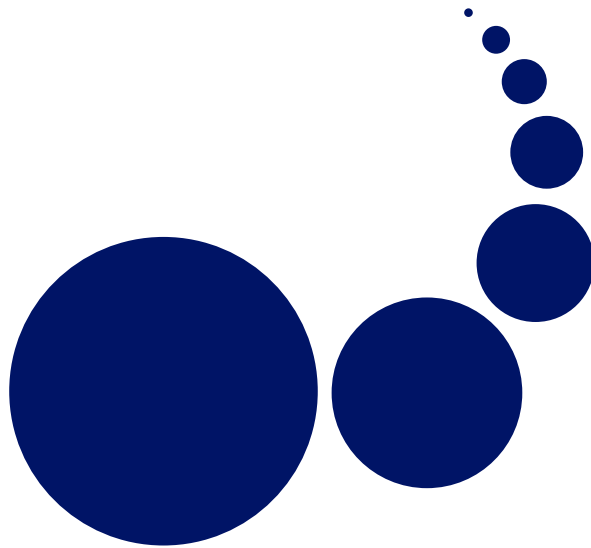


SCALABLE COMPUTING

Practice and Experience



Volume 6, Number 3, September 2005

ISSN 1895-1767



EDITOR-IN-CHIEF

Marcin Paprzycki
Institute of Computer Science
Warsaw School of Social Psychology
ul. Chodakowska 19/31
03-815 Warszawa
Poland
marcin.paprzycki@swps.edu.pl
<http://mpaprzycki.swps.edu.pl>

MANAGING EDITOR

Paweł B. Myszkowski
Institute of Applied Informatics
Wrocław University of Technology
Wyb. Wyspiańskiego 27
Wrocław 51-370, POLAND
pawel.myszkowski@pwr.wroc.pl

SOFTWARE REVIEWS EDITORS

Hong Shen
Graduate School
of Information Science,
Japan Advanced Institute
of Science & Technology
1-1 Asahidai, Tatsunokuchi,
Ishikawa 923-1292, JAPAN
shen@jaist.ac.jp

Domenico Talia
ISI-CNR c/o DEIS
Università della Calabria
87036 Rende, CS, ITALY
talia@si.deis.unical.it

TECHNICAL EDITOR

Alexander Denisjuk
Elbląg University
of Humanities and Economy
ul. Lotnicza 2
82-300 Elbląg, POLAND
denisjuk@euh-e.edu.pl

EDITORIAL BOARD

Peter Arbenz, Swiss Federal Inst. of Technology, Zürich,
arbenz@inf.ethz.ch
Dorothy Bollman, University of Puerto Rico,
bollman@cs.uprm.edu
Luigi Brugnano, Università di Firenze,
brugnano@math.unifi.it
Bogdan Czejdo, Loyola University, New Orleans,
czejdo@beta.loyno.edu
Frederic Desprez, LIP ENS Lyon,
Frederic.Desprez@inria.fr
David Du, University of Minnesota, du@cs.umn.edu
Yakov Fet, Novosibirsk Computing Center, fet@ssd.sssc.ru
Len Freeman, University of Manchester,
len.freeman@manchester.ac.uk
Ian Gladwell, Southern Methodist University,
gladwell@seas.smu.edu
Andrzej Goscinski, Deakin University, ang@deakin.edu.au
Emilio Hernández, Universidad Simón Bolívar, emilio@usb.ve
David Keyes, Old Dominion University, dkeyes@odu.edu
Vadim Kotov, Carnegie Mellon University, vkotov@cs.cmu.edu
Janusz Kowalik, Gdańsk University, j.kowalik@comcast.net
Thomas Ludwig, Ruprecht-Karls-Universität Heidelberg,
t.ludwig@computer.org
Svetozar Margenov, CLPP BAS, Sofia,
margenov@parallel.bas.bg
Oscar Naim, Oracle Corporation, oscar.naim@oracle.com
Lalit M. Patnaik, Indian Institute of Science,
lalit@micro.iisc.ernet.in
Dana Petcu, Western University of Timisoara,
petcu@info.uvt.ro
Hong Shen, Japan Advanced Institute of Science & Technology,
shen@jaist.ac.jp
Siang Wun Song, University of São Paulo, song@ime.usp.br
Bolesław Szymański, Rensselaer Polytechnic Institute,
szymansk@cs.rpi.edu
Domenico Talia, University of Calabria, talia@deis.unical.it
Roman Trobec, Jozef Stefan Institute, roman.trobec@ijs.si
Carl Tropper, McGill University, carl@cs.mcgill.ca
Pavel Tvrdik, Czech Technical University,
tvrdik@sun.felk.cvut.cz
Marian Vajtersic, University of Salzburg,
marian@cosy.sbg.ac.at
Jan van Katwijk, Technical University Delft,
J.vanKatwijk@its.tudelft.nl
Lonnie R. Welch, Ohio University, welch@ohio.edu
Janusz Zalewski, Florida Gulf Coast University,
zalewski@fgcu.edu

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

Scalable Computing: Practice and Experience

Volume 6, Number 3, September 2005

TABLE OF CONTENTS

Editorial: Challenges Concerning Symbolic Computations on Grids <i>Dana Petcu</i>	iii
Guest Editors' Introduction <i>Wilson Rivera and Jaime Seguel</i>	v
The GridWay Framework for Adaptive Scheduling and Execution on Grids <i>Eduardo Huedo, Rubén S. Montero and Ignacio M. Llorente</i>	1
Parrot: Transparent User-Level Middleware for Data-Intensive Computing <i>Douglas Thain and Miron Livny</i>	9
Satin: Simple and Efficient Java-based Grid Programming <i>Rob V. van Nieuwpoort, Jason Maassen, Thilo Kielmann and Henri E. Bal</i>	19
Run-time Adaptation of Grid Data Placement Jobs <i>G. Kola, T. Kosar and M. Livny</i>	33
JuxMem: An Adaptive Supportive Platform for Data Sharing on the Grid <i>G. Antoniu, L. Bougé and M. Jan</i>	45
Progressive Retrieval and Hierarchical Visualization of Large Remote Data <i>Hans-Christian Hege, Andrei Hutanu, Ralf Kähler, André Merzky, Thomas Radke, Edward Seidel and Brygg Ullmer</i>	57
An Adaptive File Distribution Algorithm for Wide Area Network <i>Takashi Hoshino, Kenjiro Taura and Takashi Chikayama</i>	67
Network Scheduling for Computational Grid Environments <i>Martin Swamy and Rich Wolski</i>	85
Toward Reputable Grids <i>G. von Laszewski, Beulah Kurian Alunkal and Ivana Veljkovic</i>	95
Non-Dedicated Distributed Environment: A Solution for Safe and Continuous Exploitation of Idle Cycles <i>R. C. Novaes, P. Roisenberg, R. Scheer, C. Northfleet, J. H. Jornada and W. Cirne.</i>	107



EDITORIAL: CHALLENGES CONCERNING SYMBOLIC COMPUTATIONS ON GRIDS

Symbolic and algebraic computations are currently ones of fastest growing areas of scientific computing. For a long time, the numerical approach to computational solution of mathematical problems had an advantage of being capable of solving a substantially larger set of problems than the other approach, the symbolic one. Only recently the symbolic approach gained more recognition as a viable tool for solving large-scale problems from physics, engineering or economics, reasoning, robotics or life sciences. Developments in symbolic computing were lagging relative to numerical computing, mainly due to the inadequacy of available computational resources, most importantly computer memory, but also processor power. Continuous growth in the capabilities of computer hardware led naturally to an increasing interest in symbolic calculations and resulted, among others things, in development of sophisticated Computer Algebra Systems (CASs).

CASs allow users to study computational problems on the basis of their mathematical formulations and to focus on the problems themselves instead of spending time transforming the problems into forms that are numerically solvable. While their major purpose is to manipulate formulas symbolically, many systems have substantially extended their capabilities, offering nowadays functionalities like graphics allowing a comprehensive approach to problem solving. While, typically, CAS systems are utilized in an interactive mode, in order to solve large problems they can be also used in a batch mode and programmed using languages that are close to common mathematical notation.

As CASs become capable of solving large problems, they follow the course of development that has already been taken by numerical software: from sequential computers to parallel machines to distributed computing and finally to the grid. It is particularly the grid that has the highest potential as a discovery accelerator. Currently, its widespread adoption is still impeded by a number of problems, one of which is difficulty of developing and implementing grid-enabled programs. That it is also the case for grid-enabled symbolic computations.

There are several classes of symbolic and algebraic algorithms that can perform better in parallel and distributing computing environments. For example for multiprecision integer arithmetic, that appears among others in factorizations, were developed already twenty years ago systolic algorithms and implementations on massive parallel processors, and more recently, on the Internet. Another class that utilize significant amount of computational resources is related to the implementations of polynomial arithmetic: knowledge based algorithms such as symbolic differentiation, factorization of polynomials, greatest common divisor, or, more complicated, Groebner base computations. For example, in the latest case, the size of the computation and the irregular data structures make the parallel or distributed implementation not only an attractive option for improving the algorithm performance, but also a challenge for the computational environment. A third class of algorithms that can benefit from multiple resources in parallel and distributed environments is concerning the exact solvers of large systems of equations.

The main reason driving the development of parallel and distributed algorithms for symbolic computations is the ability to solve problems that are memory bound, i.e. that cannot fit into memory of a single computer. An argument for this statement relies on the observation that the input size of a symbolic or algebraic computation can be small, but the memory used in the intermediate stages of the computation may grow considerably.

Modern CASs increase their utility not only through new symbolic capabilities, but also expanding their applicability using visualization or numerical modules and becoming more than only specific computational kernels. They are real problem solving environments based on interfaces to a significant number of computational engines. In this context it appears also the need to address the ability to reduce the wall-clock time by using parallel or distributed computing environment. A simple example is the case of rendering the images for a simulation animation.

Several approaches can be identified in the historical evolution of parallel and distributed CASs: developing versions for shared memory architectures, developing computer algebra hardware, adding facilities for communication and cooperation between existing CASs, or building distributed systems for distributed memory parallel machines or even across Internet.

Developing completely new parallel or distributed systems, although efficient, in most cases is rather difficult. Only a few parallel or distributed algorithms within such a system are fully implemented and tested. Still there are several successful special libraries and systems falling in this category: ParSac-2 system, the parallel version of SAC-2, Paclib system, the parallel extension of Saclib, FLATS based on special hardware, STAR/MPI, the parallel version of GAP, ParForm, the parallel version of Form, Cabal, MuPAD, or the recent Givaro, for parallel

computing environments, FoxBox or DSC, for distributed computing environments.

An alternative approach to build parallel and distributed CASs is to add the new value, the parallelism or the distribution, to an existing system. The number of parallel and distributed versions of most popular CASs is impressive and it can be explained by the different requirements or targeted architectures. For example, for Maple there are several implementations on parallel machines, like the one for Intel Paragon or `||Maple||`, and several implementations on networks of workstations, like Distributed Maple or PVM Maple. For Mathematica there is a Parallel Computing Toolkit, a Distributed Mathematica and a `gridMathematica` (for dedicated clusters). Matlab that provides a Symbolic Math Toolbox based on a Maple kernel has more than twenty different parallel or distributed versions: DP-Toolbox, MPITB/PVMTB, MultiMatlab, Matlab Parallelization Toolkit, ParMatlab, PMI, MatlabMPI, MATmarks, Matlab*p, Conlab, Otter and others.

More recent web-enabled systems were proved to be efficient in number theory for finding large prime numbers, factoring large numbers, or finding collisions on known encryption algorithms. Online systems for complicated symbolic computations were also built: e.g. OGB for Groebner basis computations. A framework for description and provision of web-based mathematical services was recently designed within the Monet project and a symbolic solver wrapper was built to provide an environment that encapsulates CASs and expose their functionalities through symbolic services (Maple and Axiom were chosen as computing engines). Another platform is MapleNet build on client-server architecture: the server manages concurrent Maple instances launched to server client requests for mathematical computations. WebMathematica is a similar system that offers access to Mathematica applications through a web browser.

Grid-oriented projects that involve CASs were only recent initiated. The well-known NetSolve system was one of the earliest grid system developed. Version 2 released in 2003 introduces GridSolve for interoperability with the grid based on agent technologies. APIs are available for Mathematica, Octave and Matlab. The Genss project (Grid Enabled Numerical and Symbolic Services) follows the ideas of the Monet project and intends also to combine grid computing and mathematical web services using a common agent-based framework. Several projects are porting Matlab on grids: from small ones, like Matlab*g, to very complex ones, like Geodise. Maple2g and MathGridLink are two different approaches for grid-enabled version of Maple and Mathematica. Simple to use front-end were recently build in projects like Gemlca and Websolve to deploy legacy code applications as grid services and to allows the submission of computational requests.

The vision of grid computing is that of a simple and low cost access to computing resources without artificial barriers of physical location or ownership. Unfortunately, none of the above mentioned grid-enabled CAS is responding simultaneously to some elementary requirements of a possible implementation of this vision: deploy grid symbolic services, access within CAS to available grid services, and couple different grid symbolic services.

Moreover a number of major obstacles remain to be addressed. Amongst the most important are mechanisms for adapting to dynamic changes in either computations or systems. This is especially important for symbolic computations, which may be highly irregular in terms of data and general computational demands. Such demands received until now relatively little attention from the research community.

In the context of a growing interest in symbolic computations, powerful computer algebra systems are required for complex applications. Freshly started projects shows that porting a CAS to a current distributed environment like a grid is not a trivial task not only from technological point of view but also from algorithmic point of view. Already existing tools are allowing experimental work to be initiated, but a long way is still to be cross until real-world problems will be solved using symbolic computations on grids.

Dana Petcu,
Western University of Timisoara.



GUEST EDITORS' INTRODUCTION

Grid computing focuses on building a large-scale computing infrastructure by linking computing facilities at many distributed locations. Significant effort has been spent in the design and implementation of middleware software for enabling Grid computing systems. These software packages have been successfully deployed and it is now possible to build clusters beyond the boundaries of a single local area network. However, the challenging problem of dynamically allocating resources in response to application requests for computational services remains unsolved. Adaptive middleware is software that resides between the application and the computer operating system and enables an application to adapt to changing availability of computing and networking resources. The papers for this special issue, presented for the First International Workshop on Adaptive Grid Middleware (AGridM2003), convey state-of-the-art adaptive Grid middleware and deliver important new scientific results of interest to the whole community.

Wilson Rivera,
Jaime Seguel,
University of Puerto Rico at Mayaguez.



THE GRIDWAY FRAMEWORK FOR ADAPTIVE SCHEDULING AND EXECUTION ON GRIDS*

EDUARDO HUEDO[†], RUBÉN S. MONTERO[‡], AND IGNACIO M. LLORENTE[§]

Abstract.

Many research and engineering fields, like Bioinformatics or Particle Physics, are confident about the development of Grid technologies to provide the huge amounts of computational and storage resources they require. Although several projects are working on creating a reliable infrastructure consisting of persistent resources and services, the truth is that the Grid will be a more and more dynamic entity as it grows. In this paper, we present a new tool that hides the complexity and dynamicity of the Grid from developers and users, allowing the resolution of large computational experiments in a Grid environment by adapting the scheduling and execution of jobs to the changing Grid conditions and application dynamic demands.

Key words. grid technology, bioinformatics, adaptive scheduling, adaptive execution.

1. Introduction. Grid environments inherently present the following characteristics [6]: multiple administration domains, heterogeneity, scalability, and dynamicity or adaptability. These characteristics completely determine the way scheduling and execution on Grids have to be done. For example, scalability and multiple administration domains prevent the deployment of centralized resource brokers, with total control over client requests and resource status. On the other hand, the dynamic resource characteristics in terms of availability, capacity and cost, make essential the ability to adapt job execution to these conditions.

Moreover, the emerging of Grid technology has led to a new generation of applications that relies on its own ability to adapt its execution to changing conditions [5]. These new self-adapting applications take decisions about resource selection as their execution evolves, and provide their own performance activity to detect performance slowdown. Therefore self-adapting applications can guide their own scheduling.

To deal with the dynamicity of the Grid and the adaptability of the applications two techniques has been proposed in the literature, namely:

1. *Adaptive scheduling*, to allocate pending jobs to grid resources considering the available resources, their current status, and the already submitted jobs.
2. *Adaptive execution*, to migrate running jobs to more suitable resources based on events dynamically generated by both the Grid and the application.

The AppLeS [9] project has previously dealt with the concept of *adaptive scheduling*. AppLeS is currently focused on defining templates for characteristic applications, like APST for parameter sweep and AMWAT for master/worker applications. Also, the Nimrod/G [10] resource broker dynamically optimizes the schedule to meet user-defined deadline and budget constraints. On the other hand, the need of a nomadic migration [14] approach for *adaptive execution* on a Grid environment has been previously discussed in the context of the GrADS [8] project.

In the following sections, we first explain the need for an adaptive scheduling and execution of jobs due to the dynamicity of both the Grid and the application demands. Then, in Section 3, we show a Grid-aware application model. In Section 4, we present how the GridWay framework provides support for adaptive scheduling and execution. In Section 5, we show some results obtained in the UCM-CAB research testbed with a Bioinformatics application. Finally, in Section 6, we provide some conclusions and hints about our future work.

2. Adaptive Scheduling and Execution. Grid scheduling or superscheduling [11], has been defined in the literature as the process of scheduling resources over multiple administrative domains based upon a defined policy in terms of job requirements, system throughput, application performance, budget constraints, deadlines,

*This research was supported by Ministerio de Ciencia y Tecnología (research grant TIC 2003-01321) and Instituto Nacional de Técnica Aeroespacial (INTA).

[†] Laboratorio de Computación Avanzada, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain (huedoce@inta.es).

[‡] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain (rubensm@dacya.ucm.es).

[§] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain (llorente@dacya.ucm.es) & Laboratorio de Computación Avanzada, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain (martinli@inta.es).

etc. In general, this process includes the following phases: resource discovery and selection; and job preparation, submission, monitoring, migration and termination [18].

Adaptive scheduling is the first step to deal with the dynamicity of the Grid. The schedule is re-evaluated periodically based on the available resources and their current characteristics, pending jobs, running jobs and history profile of completed jobs. Several projects [9, 10] have clearly demonstrated that periodic re-evaluation of the schedule in order to adapt it to the changing conditions, can result in significant improvements in both performance and fault tolerance.

In the case of adaptive execution, job migration is the key issue [15]. In order to obtain a reasonable degree of both application performance and fault tolerance, a job must be able to migrate among the Grid resources adapting itself to the resource availability, load (or capacity) and cost; and to the application dynamic demands.

Consequently, the following migration circumstances, related to the changing conditions and self-adapting features both discussed in Section 1, should be considered in a Grid environment:

1. *Grid-initiated migration:*

- A “better” resource is discovered (opportunistic migration [16]).
- The remote resource or its network connection fails (failover migration).
- The submitted job is canceled or suspended.

2. *Application-initiated migration:*

- Performance degradation or performance contract violation is detected in terms of application intrinsic metrics.
- The resource demands of the application change (self-migration).

The fundamental aspect of adaptive execution is the recognition of changing conditions of both Grid resources and application demands. In order to achieve such functionality, we propose a Grid-aware application model, which includes self-adapting functionality, and a *submission agent* that provides the runtime mechanisms needed to adapt the execution of the application. The application must be equipped with the functionality needed to support the application-initiated migration circumstances, while the agent is continuously watching the occurrence of the Grid- and application-initiated migration circumstances.

3. Application Model for Self-Adapting Applications. The standard application model requires modifications to be Grid-aware. In the following list (see figure 3.1) we detail the extension of the classical application paradigm in order to take advantage of the Grid capabilities and to be aware of its dynamic conditions:

- A **requirement expression** is necessary to specify the application requirements that must be met by the target resources. This file can be subsequently updated by the application to adapt its execution to its dynamic demands. The application could define an initial set of requirements and dynamically change them when more, or even less, resources are required.
- A **ranking expression** is necessary to dynamically assign a rank to each resource, in order to prioritize the resources that fulfill the requirements according to the application runtime needs. A compute-intensive application would assign a higher rank to those hosts with faster processors and lower load, while a data-intensive application could benefit those hosts closer to the input data [16].
- A **performance profile** is advisable to keep the application performance activity in terms of application intrinsic metrics, in order to detect performance slowdown. For example, it could maintain the time consumed by the code in the execution of a set of given fragments, in each cycle of an iterative method or in a set of given input/output operations.

Due to the high fault rate and the dynamic rescheduling, **restart files** are highly advisable. Migration is commonly implemented by restarting the job on the new candidate host, so the job should generate restart files at regular intervals in order to restart execution from a given point. However, for some application domains the cost of generating and transferring restart files could be greater than the saving in compute time due to checkpointing. Hence, if the checkpointing files are not provided the job should be restarted from the beginning. User-level checkpointing managed by the programmer must be implemented because system-level checkpointing is not possible among heterogeneous resources.

The application source code does not have to be modified if the application is not required to be self-adaptive. However, our infrastructure requires changing the source code or inserting instrumentation instructions in compiled code when the application takes decisions about resource selection and provides its own performance activity.

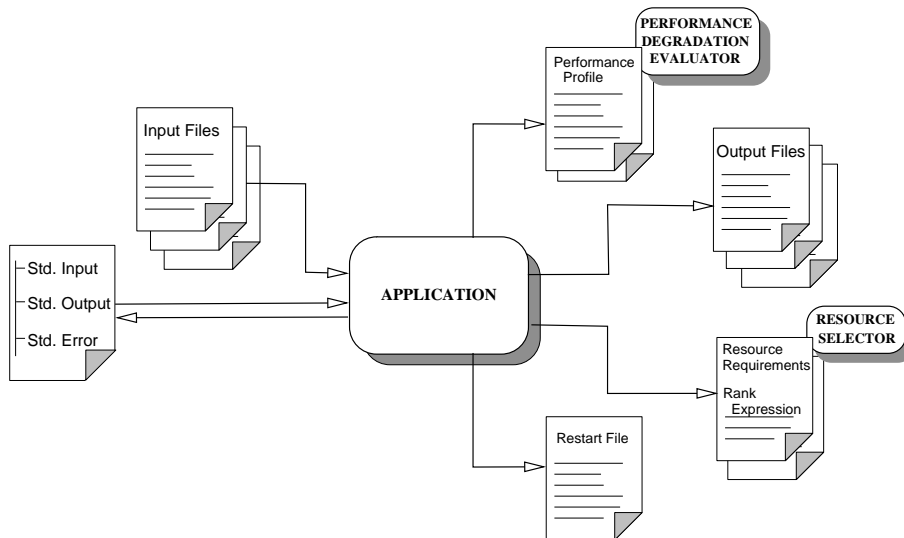


FIG. 3.1. Model for self-adapting applications.

With self-adapting capabilities, an application could initially define a minimal set of requirements and, after it begins to run, it can change them to a more restricted set. In this way, the application will have more chances to find a resource to run on, and once running, it will migrate only if the candidate resource worths it.

Note also that if the application is divided in several phases, each one with different requirements, it could change them progressively to be more or less restrictive. In this way, the application does not have to impose the most restricted set of requirements at the beginning, since it limits the chance for the application to begin execution (see Section 5.3.2). Moreover, the application have the choice to make the requirement change optional or mandatory, i.e. it can check if the current resource meets the new requirements, otherwise it may request a (self-)migration.

4. GridWay Support for Adaptive Scheduling and Execution. GridWay is a new experimental framework based on Globus [4] that allows an easier and more efficient execution of jobs on a dynamic Grid environment in a “submit and forget” fashion. The core of the GridWay framework [13] is a personal *submission agent* that performs all the scheduling stages [18] and watches over the correct and efficient execution of jobs. Adaptation to changing conditions is achieved by dynamic rescheduling: once the job is initially allocated, it is rescheduled when a migration circumstance (discussed in Section 2) is detected.

Job execution is performed in three stages by the following modules, which can be defined on a per job basis:

- The *prolog* module, which prepares the remote system and stages the input files.
- The *wrapper* module, which executes the actual job and returns its exit code.
- The *epilog* module, which stages the output files and cleans up the remote system.

Migration is performed by combining the above stages. First, the *wrapper* is canceled (if it is still running), then the *prolog* is submitted to the new candidate resource, preparing it and transferring to it all the needed files, including the `restart` files from the old resource. After that, the *epilog* is submitted to the old resource (if it is still available), but no output file staging is performed, it only cleans up the remote system. Finally, the *wrapper* is submitted to the new candidate resource.

The *submission agent* uses the following modules, which also can be defined on a per job basis, to provide the application with the support needed for implementing self-adapting functionality:

- The *resource selector* module, which evaluates the `requirement` and `ranking` expressions when the job has to be scheduled or rescheduled. Different strategies for resource selection can be implemented, from the simplest one based on a pre-defined list of hosts to more advanced strategies based on requirement filtering, and resource ranking in terms of performance models.
- The *performance evaluator* module, which periodically evaluates the application’s `performance profile` in order to detect performance slowdown and so request a rescheduling action.

Different strategies could be implemented, from the simplest one based on querying the Grid information services about system status information to more advanced strategies based on detection of performance contract violations.

The *submission agent* also provides the application with the fault tolerance capabilities needed in such a faulty environment:

- The GRAM [1] *job manager* notifies submission failures as GRAM callbacks. This kind of failures includes, among others, connection, authentication, authorization, RSL parsing, executable or input staging, credential expiration. . .
- The *job manager* is probed periodically at each *polling* interval. If the *job manager* does not respond, the GRAM *gatekeeper* is probed. If the *gatekeeper* responds, a new *job manager* is started to resume watching over the job. If the *gatekeeper* fails to respond, a resource or network occurred. This is the approach followed by Condor-G [12].
- The standard output of *prolog*, *wrapper* and *epilog* is parsed in order to detect failures. In the case of the *wrapper*, this is useful to capture the job exit code, which is used to determine whether the job was successfully executed or not. If the job exit code is not set, the job was prematurely terminated, so it failed or was intentionally canceled.

When an unrecoverable failure is detected, the *submission agent* retries the submission of *prolog*, *wrapper* or *epilog* a number of times specified by the user and, when no more retries are left, it performs an action chosen by the user among two possibilities: stop the job for manually resuming it later, or automatically reschedule it.

We have developed both an API (subset of the DRMAA [17] standard proposed in the GGF [3]) and a command line interface to interact with the *submission agent*. They allow scientists and engineers to express their computational problems in a Grid environment. The capture of the remote execution exit code allow users to define complex jobs, where each depends on the output and exit code from the previous job. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the work flow of certain type of applications.

Our framework is not bounded to a specific class of applications, does not require new services, and does not necessarily require source code changes. The framework is currently functional on any Grid testbed based on Globus. We believe that is an important advantage because of socio-political issues: cooperation between different organizations, administrators, and users can be very difficult.

5. Experiences.

5.1. The Target Application. We have tested our tool with a Bioinformatics application aimed at predicting the structure and thermodynamic properties of a target protein from its amino acid sequences. The algorithm, tested in the 5th round of Critical Assessment of techniques for protein Structure Prediction (CASP5), aligns with gaps the target sequence with all the 6150 non-redundant structures in the Protein Data Bank (PDB), and evaluates the match between sequence and structure based on a simplified free energy function plus a gap penalty term. The lowest scoring alignment found is regarded as the prediction if it satisfies some quality requirements. For each sequence-structure pair, the search of the optimal alignment is not exhaustive. A large number of alignments are constructed in parallel through a semi-deterministic algorithm, which tries to minimize the scoring function.

To speed up the analysis and reduce the data needed, the PDB files are preprocessed to extract the contact matrices, which provide a reduced representation of protein structures. The algorithm is then applied twice, the first time as a fast search, in order to select the 100 best candidate structures, the second time with parameters allowing a more accurate search of the optimal alignment.

We have applied the algorithm to the prediction of thermodynamic properties of families of orthologous proteins, i.e. proteins performing the same function in different organisms. If a representative structure of this set is known, the algorithm predicts it as the correct structure. The biological results of the comparative study of several proteins are presented elsewhere [19, 7].

5.2. Experiment Preparation. We have modified the application to provide a `restart file` and a `performance profile`. The architecture independent `restart file` stores the best candidate proteins found to that moment and the next protein in the PDB to analyze. The `performance profile` stores the time spent on each iteration of the algorithm, where an iteration consists in the analysis of a given number of sequences.

TABLE 5.1
The UCM-CAB research testbed.

Name	Architecture	OS	Speed	Memory	Job mgr.	VO
ursa	1×UltraSPARC-IIe	Solaris	500MHz	256MB	fork	UCM
draco	1×UltraSPARC-I	Solaris	167MHz	128MB	fork	UCM
pegasus	1×Pentium 4	Linux	2.4GHz	1GB	fork	UCM
solea	2×UltraSPARC-II	Solaris	296MHz	256MB	fork	UCM
babieca	5×Alpha EV6	Linux	466MHz	256MB	PBS	CAB

Initially, the application does not impose any requirement to the resources, so the `requirement expression` is null. The `ranking expression` uses a performance model to estimate the job turnaround time as the sum of execution and transfer time, derived from the performance and proximity of the candidate resources [16].

The `resource selector` consists of a shell script that queries the MDS [2] for potential execution hosts. Initially, available compute resources are discovered by accessing the GIIS server and those resources that do not meet the user-provided requirements are filtered out. At this step, an authorization test (via GRAM ping request) is performed on each discovered hosts to guarantee user access. Then, the resource is monitored to gather its dynamic status by accessing its local GRIS server. This information is used to assign a rank to each candidate resource based on user-provided preferences. Finally, the resultant prioritized list of candidate resources is used to dispatch the jobs.

In order to reduce the information retrieval overhead, the GIIS and GRIS information is locally cached at the client host and updated independently in order to separately determine how often the testbed is searched for new resources and the frequency of resource monitoring. In the following experiments we set the GIIS cache timeout to 5 minutes and the GRIS cache timeout to 30 seconds.

The `performance evaluator` is another shell script that parses the `performance profile` and detects performance slowdown when the last iteration time is greater than a given threshold.

The whole experiment was submitted as an array job, where each sequence was analyzed in a separate task of the array, specifying all the needed information in a `job template` file.

The experiment files consists of: the executable (0.5MB) provided for all the resource architectures in the testbed, the PDB files shared and compressed (12.2MB) to reduce the transfer time, the parameter files (1KB), and the file with the sequence to be analyzed (1KB). The final file name of the executable and the file with the sequence to be analyzed is obtained by resolving the variables `GW_ARCH` and `GW_TASK_ID`, respectively, at runtime for the current host and job. Input files can be local or remote (specified as a GASS or GridFTP URL), and both can be compressed (to be uncompressed on the selected host) and declared as shared (then stored in the GASS cache and shared by all the jobs submitted to this resource).

5.3. Results on the UCM-CAB Testbed. We have performed the experiments in the UCM-CAB research testbed, which is summarized in table 5.1.

5.3.1. Detection of a Performance Degradation. Let us first consider an experiment consisting in five tasks, each of them applies the structure prediction algorithm to a different sequence of the *ATP Synthase* enzyme (epsilon chain) present in different organisms. Shortly after submitting the experiment, `pegasus` was overloaded with a compute-intensive application.

Figure 5.1 shows the execution profile in this situation, along with the load in `pegasus` that caused the performance degradation, and the progress of job 0, obtained from its `performance profile`. Initially four tasks are allocated to `babieca` and one to `pegasus`. When the `performance evaluator` detects the performance degradation, it requests a job migration. Since there is a slot available in `babieca`, the job is migrated to it although it presents lower performance. In spite of the overhead induced by job migration, 6% of the total execution time, job 0 ends before the rest of jobs, because of the better performance offered by `pegasus` before it became saturated.

5.3.2. Mandatory Change in Resource Requirements. In the following experiment, we have applied the structure prediction algorithm to five sequences of the *Triosephosphate Isomerase* enzyme, which is considerably larger than the previous one, present in different organisms.

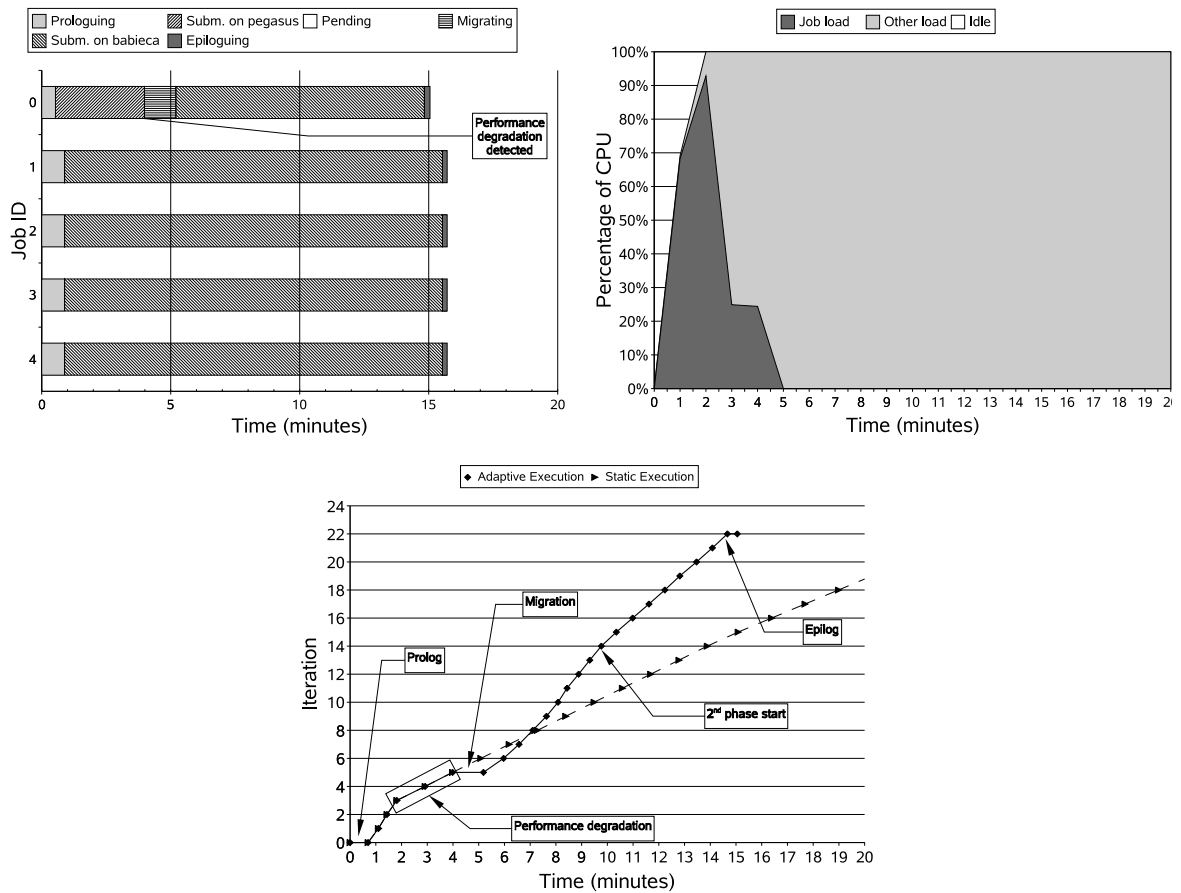


FIG. 5.1. Execution profile (top), load in pegasus (middle), and progress of job 0 (bottom) when a performance degradation is detected.

As mentioned in Section 5.1, the target application is divided in two different phases. First, a fair analysis is performed to get the 100 best candidate proteins, and then, a more exhaustive analysis is performed to get the 20 best candidate proteins from the 100 obtained in the first phase. As the second phase analysis performs a more accurate sequence alignment and the target sequence is quite large, it needs more memory than the first phase analysis. Therefore, the application change its resource requirements before starting the second phase to assure that it has enough memory (512MB). The only resource that meets the requirements of the second phase is pegasus.

Figure 5.2 shows the execution profile in this situation. Job 0 starts execution on pegasus, while jobs 1 to 4 start execution on babieca. When job 0 completes its execution, job 1 detects that pegasus has become free and migrates to it, since it presents a better rank (opportunistic job migration). After that, jobs 2 to 4 request a self-migration as they have changed their requirements to complete the second phase of the protein analysis and babieca doesn't meet them. Jobs 0 and 1 also changed their requirements before, but its execution host in that moment (pegasus) met them, so they could continue with their execution. As pegasus is busy with job 1, jobs 2 to 4 have to wait until it becomes available. These jobs are submitted consecutively to pegasus (see figure 5.2) to complete the second phase of the protein analysis.

6. Conclusions. We have shown an effective way for providing adaptive scheduling and execution on Grids. The presented framework does not necessarily require source code changes in the applications, but with minimal changes, applications could benefit from the self-adapting features also provided.

On the scope of the target application, these promising experiments show the potentiality of the Grid to the study of large numbers of protein sequences, and suggests the possible application of this methods to the whole set of proteins in a complete microbial genome.

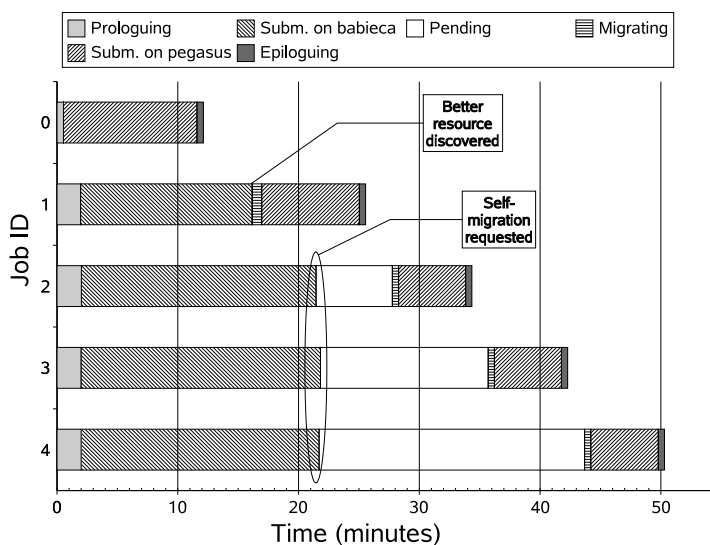


FIG. 5.2. Execution profile when a mandatory change in resource requirements occurs.

We are currently working on a *storage resource selector* module to provide support for replica files, specified as a logical file or as a file belonging to a logical collection. In this way the PDB files holding the protein structures, will be scattered on the Grid testbed. The discovery process is performed by accessing the Globus Replica Catalog. The resource selection is based on the proximity between the selected compute resource and the candidate storage resources, along with the values gathered from the MDS GRIS.

Acknowledgments. We would like to thank Ugo Bastolla, staff scientist at the Centro de Astrobiología and developer of the Bioinformatics application used in the experiments, for his support on understanding and modifying the application.

REFERENCES

- [1] *Globus Resource Allocation Manager*. <http://www.globus.org/gram>.
- [2] *Monitoring and Discovery Service*. <http://www.globus.org/mds>.
- [3] *The Global Grid Forum*. <http://www.gridforum.org>.
- [4] *The Globus Project*. <http://www.globus.org>.
- [5] G. ALLEN, E. SEIDEL, AND J. SHALF, *Scientific Computing on the Grid*, Byte, Spring 2002 (2002), pp. 24–32.
- [6] M. BAKER, R. BUYYA, AND D. LAFORENZA, *Grids and Grid Technologies for Wide-Area Distributed Computing*, Intl. J. of Software: Practice and Experience (SPE), 32 (2002), pp. 1437–1466.
- [7] U. BASTOLLA ET AL., *Reduced Protein Folding Efficiency, Genome Reduction and AT Bias in Obligatory Intracellular Bacteria: An Integrated View*, (2003). (preprint).
- [8] F. BERMAN ET AL., *The GrADS Project: Software Support for High-Level Grid Application Development*, Intl. J. of High Performance Computing Applications, 15 (2001), pp. 327–34.
- [9] ———, *Adaptive Computing on the Grid Using AppLeS*, IEEE Transactions on Parallel and Distributed Systems, 14 (2003), pp. 369–382.
- [10] R. BUYYA, D. ABRAMSON, AND J. GIDDY, *A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker*, Future Generation Computer Systems, 18 (2002), pp. 1061–1074.
- [11] I. FOSTER AND C. KESSELMAN, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufman, 1999.
- [12] J. FREY ET AL., *Condor/G: A Computation Management Agent for Multi-Institutional Grids*, in Proc. of the 10th Symp. on High Performance Distributed Computing (HPDC10), 2001.
- [13] E. HUEDO, R. S. MONTERO, AND I. M. LLORENTE, *A Framework for Adaptive Execution on Grids*, Intl. J. of Software – Practice and Experience, (2004). (in press).
- [14] G. LANFERMANN ET AL., *Nomadic Migration: A New Tool for Dynamic Grid Computing*, in Proc. of the 10th Symp. on High Performance Distributed Computing (HPDC10), 2001.
- [15] R. S. MONTERO, E. HUEDO, AND I. M. LLORENTE, *Experiences about Job Migration on a Dynamic Grid Environment*, in Proc. of Intl. Conf. on Parallel Computing (ParCo 2003), September 2003.
- [16] ———, *Grid Resource Selection for Opportunistic Job Migration*, in Proc. of Intl. Conf. on Parallel and Distributed Computing (Euro-Par 2003), vol. 2790 of Lecture Notes on Computer Science, August 2003, pp. 366–373.

- [17] H. RAJIC ET AL., *Distributed Resource Management Application API Specification 1.0*, tech. rep., The Global Grid Forum, 2003. DRMAA Working Group.
- [18] J. M. SCHOPF, *Ten Actions when Superscheduling*, Tech. Rep. GFD-I.4, The Global Grid Forum: Scheduling Working Group, 2001.
- [19] R. VAN HAM ET AL., *Reductive Genome Evolution in buchnera aphidicola*, Proc. Natl. Acad. Sci. USA, 100 (2003), pp. 581–586.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 3, 2003.

Accepted: September 1, 2003.



PARROT: AN APPLICATION ENVIRONMENT FOR DATA-INTENSIVE COMPUTING

DOUGLAS THAIN AND MIRON LIVNY*

Abstract. Distributed computing continues to be an alphabet-soup of services and protocols for managing computation and storage. To live in this environment, applications require middleware that can transparently adapt standard interfaces to new distributed systems; such middleware is known as an interposition agent. In this paper, we present several lessons learned about interposition agents via a progressive study of design possibilities. Although performance is an important concern, we pay special attention to less tangible issues such as portability, reliability, and compatibility. We begin with a comparison of seven methods of interposition and select one method, the debugger trap, that is the slowest but also the most reliable. Using this method, we implement a complete interposition agent, Parrot, that splices existing remote I/O systems into the namespace of standard applications. The primary design problem of Parrot is the mapping of fixed application semantics into the semantics of the available I/O systems. We offer a detailed discussion of how errors and other unexpected conditions must be carefully managed in order to keep this mapping intact. We conclude with an evaluation of the performance of the I/O protocols employed by Parrot, and use an Andrew-like benchmark to demonstrate that semantic differences have consequences in performance.¹

Key words. Adaptive middleware, error diagnosis, interposition agents, virtual machines.

1. Introduction. The field of distributed computing has produced countless systems for harnessing remote processors and accessing remote data. Despite the intentions of their designers, no single system has achieved universal acceptance or deployment. Each carries its own strengths and weakness in performance, manageability, and reliability. Renewed interest in world-wide computational systems is increasing the number of protocols and interfaces in play. A complex ecology of distributed systems is here to stay.

The result is an hourglass model of distributed computing, shown in Figure 1.1. At the center lie ordinary applications built to standard interfaces such as POSIX. Above lie a number of batch systems that manage processors, interact with users, and deal with failures of execution. A batch system interacts with an application through simple interfaces such as *main* and *exit*. Below lie a number of I/O services that organize and communicate with remote memory, disks, and tapes. An ordinary operating system (OS) transforms an application's explicit *reads* and *writes* into the low-level block and network operations that compose a local or distributed file system.

However, attaching a new I/O service to a traditional OS is not a trivial task. Although the principle of an extensible OS has received much attention in the research community [19], production operating systems have limited facilities for extension, usually requiring kernel modifications or administrator privileges. Although this may be acceptable for a personal computer, this requirement makes it difficult or impossible to provide custom I/O and naming services for applications visiting a borrowed computing environment such as a timeshared mainframe, a commodity computing cluster, or an opportunistic workgroup.

To remedy this situation, we advocate the use of *interposition agents* [13]. These devices transform standard interfaces into remote I/O protocols not normally found in an operating system. In effect, an agent allows an application to bring its filesystem and namespace along with it wherever it goes. This releases the dependence on the details of the execution site while preserving

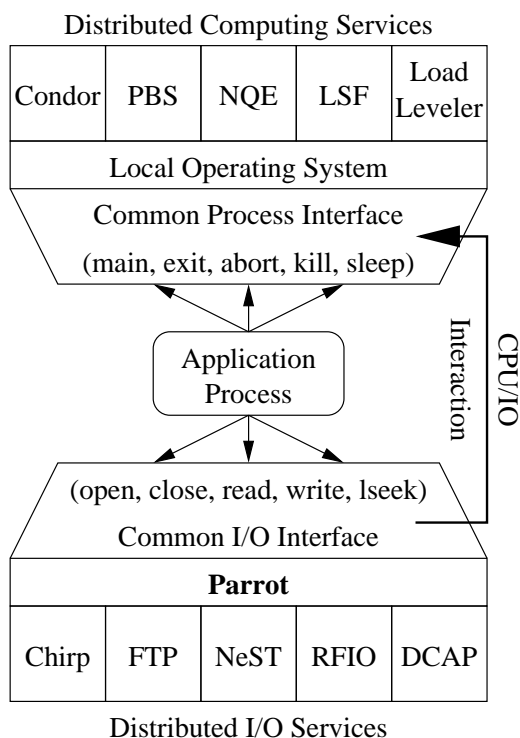


FIG. 1.1. The Hourglass Model

the use of standard interfaces. In addition, the agent can tap into naming services that transform private names into fully-qualified names relevant in the larger system.

*Computer Sciences Department, University of Wisconsin

¹This research was supported by a Lawrence Landweber NCR fellowship in distributed systems.

	internal techniques				external techniques		
	poly. exten.	static link	dyn. link	binary rewrite	debug trap	remote filesys.	kernel callout
scope	library	static	dynamic	dynamic	no setuid	any	any
burden	rewrite	relink	identify	identify	run command	superuser	modify os
layer	fixed	any	any	any	syscall	fs ops only	syscall
init/fini	hard	hard	hard	hard	easy	impossible	easy
aff. linker	no	no	no	no	yes	yes	yes
debug	yes	yes	yes	yes	limited	yes	yes
secure	no	no	no	no	yes	yes	yes
find holes	easy	hard	hard	hard	easy	easy	easy
porting	easy	hard	hard	hard	medium	easy	medium

FIG. 1.2. *Properties of Interposition Techniques*

In this paper, we present practical lessons learned from several years of building and deploying interposition agents within the Condor project. [20, 28, 21, 22] Although the notion of such agents is not unique to Condor [13, 2, 12], they have seen relatively little use in other production systems. This is due to a variety of technical and semantic difficulties that arise in connecting real systems together.

We present this paper as a progressive design study that explores these problems and explains our solutions. We begin with a detailed study of seven methods of interposition, five of which we have experience building and deploying. The remaining two are effective but impractical because of the privilege required. We will compare the performance and functionality of these methods, giving particular attention to intangibles such as portability and reliability. In particular, we will concentrate on one method that has not been explored in detail: the debugger trap. Although this method has been employed in idealized operating systems, it requires additional techniques in order to provide acceptable performance on popular operating systems with limited debugging capabilities, such as Linux.

Using the debugger trap, we focus on the design of Parrot, an interposition agent that splices remote I/O systems into the filesystem space of ordinary applications. A central problem in the design of an I/O agent is the semantic problem of mapping not-quite-identical interfaces to each other. The outgoing mapping is usually quite simple: *read* becomes a *get*, *write* becomes a *put*, and so forth. The real difficulty lies in interpreting the large space of return values from remote services. Many new kinds of failure are introduced: servers crash, credentials expire, and disks fill. Trivial transformations into the application’s standard interface lead to a brittle and frustrating experience for the user.

A corollary to this observation is that access to computation and storage cannot be fully divorced. Abstract notions of design often encourage the partition of distributed systems into two activities: either computation or storage. An interposition agent serves as a connection between these two concerns; like an operating system kernel, it manages both types of devices and must mediate their interaction, sometimes bypassing the application itself.

This paper is a condensed version of a workshop paper. Due to space limitations, we have omitted a number of sections and details, indicated by footnotes. The interested reader may find further details in the original paper [23] or in a technical report. [24]²

2. Interposition Techniques Compared. There are many techniques for interpositioning services between an application and the underlying system. Each has particular strengths and weaknesses. Figure 1.2 summarizes seven interposition techniques. They may be broken into two broad categories: internal and external. Internal techniques modify the memory space of an application process in some fashion. These techniques are flexible and efficient, but cannot be applied to arbitrary processes. External techniques capture and modify operations that are visible outside an application’s address space. These techniques are less flexible and have higher overhead, but can be applied to nearly any process. The Condor project has experience building and deploying all of the internal techniques as well one external technique: the debugger trap. The remaining two external techniques we describe from relevant publications.

The simplest technique is the *polymorphic extension*. If the application structure is amenable to extension, we may simply add a new implementation of an existing interface. The user then must make small code changes to invoke the appropriate constructor or factory in order to produce the new object. This technique is used in

²Omitted: Example applications of interposition agents.

Condor’s Java Universe [22] to connect an ordinary `InputStream` or `OutputStream` to a secure remote proxy. It is also found in general purpose libraries such as SFIO [25].

The *static library* technique involves creating a replacement for an existing library. The user is obliged to re-link the application with the new library. For example, Condor’s Standard Universe [20] provides a drop-in replacement for the standard C library that provides transparent checkpointing as well as proxying of I/O back to the submission site, fully emulating the user’s home environment. The *dynamic library* technique also involves creating a replacement for an existing library. However, through the use of linker controls, the user may direct the new library to be used in place of the old for any given dynamically linked library. This technique is used by DCache [8], some implementations of SOCKS [15], as well as our own Bypass [21] toolkit. The *binary rewriting* technique involves modifying the machine code of a process at runtime to redirect the flow of control. This requires very detailed knowledge of the CPU architecture in use, but this can be hidden behind an abstraction such as the Paradyn [17] toolkit. This technique has been used to “hijack” an unwitting process at runtime [28].

Traditional debuggers make use of a specialized operating system interface for stopping, examining, and resuming a process. The *debugger trap* technique uses this interface, but instead of merely examining the process, the debugging agent traps each system call, provides an implementation, and then places the result back in the target process while nullifying the intended system call. An example of this technique is UFO [2], which allows access to HTTP and ftp resources via whole-file fetching. A difficulty with the debugger trap is that many tools compete for access to a single process’ debug interface. The Tool Daemon Protocol (TDP) [18] provides an interface for managing such tools in a distributed system.

A *remote filesystem* may be used as an interposition agent by simply modifying the file server. NFS is a popular choice for this technique, and is used by the Legion [27] object-space translator, as well the Slice [4] microproxy. Finally, short of modifying the kernel itself, we may install a one-time *kernel callout* which permits a filesystem to be serviced by a user-level process. This facility can be present from the ground up in a microkernel [1], but can also be added as an afterthought, which is the case for most implementations of AFS [11].

The four internal techniques may only be applied to certain kinds of programs. Polymorphic extension and static linking only apply to those programs that can be rebuilt. The dynamic library technique requires that the replaced library be dynamic, while binary rewriting (with the Paradyn toolkit) requires the presence of the dynamic loader, although no particular library must be dynamic. The three external techniques apply to any process, with the exception that the debugging trap prevents the traced process from elevating its privilege level through the *setuid* feature.

The burden upon the user for each of these techniques also varies widely. For example, polymorphic extension requires small code changes while static linking requires rebuilding. These techniques may not be possible with packaged commercial software. Dynamic linking and binary rewriting require that the user understand which programs are dynamically linked and which are not. Most standard system utilities are dynamic, but many commercial packages are static. Our experience is that users are surprised and quite frustrated when an (unexpectedly) static application blithely ignores an interposition agent. The remote filesystem and kernel callout techniques impose the smallest user burden, but require a cooperative system administrator to make the necessary changes. The debugger trap imposes a small burden on the user to simply invoke the agent executable.

Perhaps the most significant difference between the techniques is the ability to trap different layers of software. Each of the internal techniques may be applied at any layer of code. For example, Bypass has been used to instrument an application’s calls to the standard memory allocator, the X Window System library, and the OpenGL library. In contrast, the external techniques are fixed to particular interfaces. The debugger trap only operates on physical system calls, while the remote filesystem and kernel callout are limited to certain filesystem operations.

Differences in these techniques affect the design of code that they attach to. Consider the matter of implementing a directory listing on a remote device. The internal techniques are capable of intercepting library calls such as *open* and *opendir*. These are easily mapped to remote file access protocols, which generally have separate procedures for accessing files and directories. However, the Unix interface unifies files and directories; both are accessed through the system call *open*. External techniques must accept an *open* on either a file or directory and defer the binding to a remote operation until either *read* or *getdents* is invoked. The choice of interposition layer affects the design of the agent.

The external techniques also differ in the range of operations that they are able to trap. While the debugger trap can modify any system call, the remote filesystem and kernel callout techniques are limited to filesystem operations. A particular remote filesystem may have even further restrictions. For example, the stateless NFS protocol has no representation of the system calls *open* and *close*. Without access to this information, the interposed service cannot provide semantics significantly different than those provided by NFS. Further, such file system interfaces do not express any binding between individual operations and the processes that initiate them. That is, a remote filesystem agent sees a *read* or *write* but not the process id that issued it. Without this information, it is difficult or impossible to performing accounting for the purposes of security or performance.

A number of important activities take place during the initialization and finalization of a process: dynamic libraries are loaded; constructors, destructors, and other automatic routines are run; I/O streams are created or flushed. During these transitions, the libraries and other resources in use by a process are in a state of flux. This complicates the implementation of internal agents that wish to intercept such activity. For example, the application may perform I/O in a global constructor or destructor. Thus, an internal agent itself cannot rely on global constructors or destructors: there is no ordering enforced between those of the application and those of the agent. Likewise, a dynamically loaded agent cannot interpose on the actions of the dynamic linker. The programmer of such agents must not only exercise care in constructing the agent, but also in selecting the libraries invoked by the agent. Such code is time consuming to create and debug. These activities are much more easily manipulated through external techniques. For example, external techniques can easily trap and modify the activities of the dynamic linker.

No code is ever complete nor fully debugged. Production deployment of interposition agents requires that users be permitted to debug both applications and agents. All techniques admit debugging of user programs, with the only complication arising in the debugger trap. For obvious reasons, a single process cannot be debugged by two processes at once, so a debugger cannot be attached to an instrumented process. However, a debugger trap agent can be used to manage an entire process tree, so instead the user may use the agent to invoke the debugger, which may then invoke the application. The debugger's operations may be trapped just like any other system call and passed along to the application, all under the supervision of the agent.

Interposition agents may be used for security as well as convenience. An agent may provide a *sandbox* which prevents an untrusted application from modifying any external data that it is not permitted to access. The internal techniques are not suitable for this security purpose, because they may easily be subverted by a program that invokes system calls directly without passing through libraries. The external techniques, however, cannot be fooled in this way and are thus suitable for security.

Related to security is the matter of *hole detection*. An interposition agent may fail to trap an operation attempted by an application. This may simply be a bug in the agent, or it may be that the interface has evolved over time, and the application is using a deprecated or newly added interface that the agent is not aware of. Internal agents are especially sensitive to this bug. As standard libraries develop, interfaces are added and deleted, and modified library routines may invoke system calls directly without passing through the corresponding public interface function. For example, *fopen* may invoke the *open* system call without passing through the *open* function. Such an event causes general chaos in both the application and agent, often resulting in crashes or (worse) silent output errors. No such problem occurs in external agents. Although interfaces still change, any unexpected event is detected as an unknown system call. The agent may then terminate the application and indicate the exact problem.

The problem of hole detection must not be underestimated. Our experience is that any significant operating system upgrade includes changes to the standard libraries, which in turn require modifications to internal trapping techniques. Thus, internal agents are rarely forward compatible. Further, identifying and fixing such holes is time consuming. Because the missed operation itself is unknown, one must spend long hours with a debugger to see where the expected course of the application differs from the actual behavior. Once discovered, a new entry point must be added to the agent. The treatment is simple but the diagnosis is difficult. We have learned this lesson the hard way by porting both the Condor remote system call library and the Bypass toolkit to a wide variety of Unix-like platforms.

For these reasons, we have described *porting* in Figure 1.2 as follows. The polymorphic extension and the remote filesystem are quite easy to build on a new system. The debugger trap and the kernel callout have significant system dependent components to be ported to each operating system, but the nature and stability of these interfaces make this a tractable task. The remaining three techniques—static linking, dynamic linking,

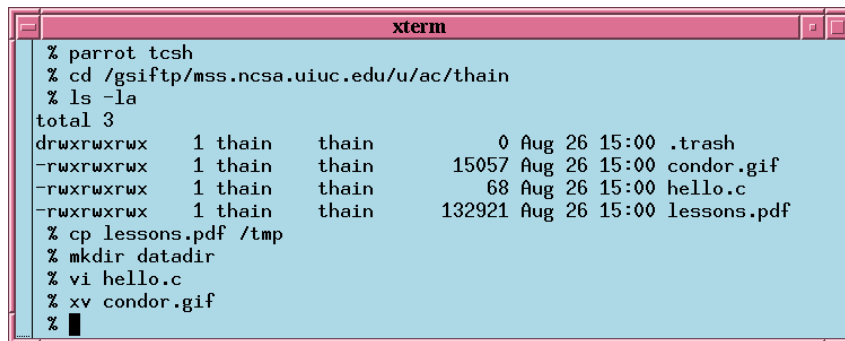
	getpid	stat	open/close	read 8KB	bandwidth
unmod	.18±.03 μ s	1.85±.09	3.18± .08	3.27± .19	282±13 MB/s
rewrite	.21±.25 μ s	1.82±.02	3.21± .05	3.26± .03	280± 7 MB/s
static	.21±.02 μ s	1.80±.17	3.59± .05	3.34± .02	280±17 MB/s
dynamic	1.22±.01 μ s	3.60±.10	5.53± .06	4.31± .09	278± 4 MB/s
(α unmod)	(6.8x)	(1.9x)	(1.7x)	(1.3x)	(0.99x)
debug	10.06±.21 μ s	55.41±.50	42.09± .06	30.99± .26	122± 4 MB/s
(α unmod)	(56x)	(30x)	(13x)	(9x)	(0.43x)

FIG. 2.1. *Overhead of Interposition Techniques*

and binary rewriting—should be viewed as a significant porting challenge that must be revisited at every minor operating system upgrade.

Figure 2.1 compares the performance of four transparent interposition techniques. We constructed a benchmark C program which timed 100,000 iterations of various system calls on a 1545 MHz Athlon XP1800 running Linux 2.4.18. Available bandwidth was measured by reading a 100 MB file sequentially in 1 MB blocks. The mean and standard deviation of 1000 cycles of each benchmark are shown. File operations were performed on an existing file in a temporary file system. The *unmod* case gives the performance of this benchmark without any agent attached, while the remaining five show the same benchmark modified by each interposition technique. In each case, we constructed a very minimal agent to trap system calls and invoke them without modification.

As can be seen, the binary rewriting and static linking methods add no significant cost to the application. The dynamic method has overhead on the order of microseconds, as it must manage the structure of (potentially) multiple agents and invoke a function pointer. However, these overheads are quickly dominated by the cost of moving data in and out of the process. The debugger trap has the greatest overhead of all the techniques, ranging from a 56x slowdown for *getpid* to a 6x slowdown for writing 8 KB. Most importantly, the bandwidth measurement demonstrates that the debugger trap achieves less than half of the unmodified I/O bandwidth. It should be fairly noted that this latency and bandwidth will be dominated by the latency and bandwidth of accessing remote services on commodity networks. Security and reliability come at a measurable cost.³



```

xterm
% parrot tcsh
% cd /gsiftp/mss.ncsa.uiuc.edu/u/ac/thain
% ls -la
total 3
drwxrwxrwx  1 thain  thain          0 Aug 26 15:00 .trash
-rwxrwxrwx  1 thain  thain       15057 Aug 26 15:00 condor.gif
-rwxrwxrwx  1 thain  thain         68 Aug 26 15:00 hello.c
-rwxrwxrwx  1 thain  thain    132921 Aug 26 15:00 lessons.pdf
% cp lessons.pdf /tmp
% mkdir datadir
% vi hello.c
% xv condor.gif
% █

```

FIG. 3.1. *Interactive Browsing with Parrot*

3. Parrot. The Parrot interposition agent attaches standard applications to a variety of distributed I/O systems by way of the debugger trap, described above. Each I/O protocol is presented as a normal filesystem entry under a new top-level directory bearing the name of the protocol. In addition, an optional *mountlist* may be given, which redirects parts of the filesystem namespace to external paths. Figure 3.1 shows Parrot being used with standard tools to manipulate files stored at the Mass Storage Server (MSS) at the National Center for Supercomputing Applications (NCSA) via the Grid Security Infrastructure (GSI) [9] variant of the File Transfer Protocol (FTP).

Parrot is equipped with a variety of drivers for communicating with external storage systems; each has particular features and limitations. The simplest is the **Local** driver, which simply passes operations on to the underlying operating system. The **Chirp** protocol was designed by the authors in an earlier work [22]

³Omitted: a detailed description of the debugger trap.

to provide remote I/O with semantics very similar to POSIX. A standalone chirp server is distributed with Parrot. The venerable **File Transfer Protocol (FTP)** has been in heavy use since the early days of the Internet. Its simplicity allows for a wide variety of implementations, which, for our purposes, results in an unfortunate degree of imprecision which we will expand upon below. Parrot supports the secure GSI [3] variant of ftp. The **NeST** protocol is the native language of the NeST storage appliance [6], which provides an array of authentication, allocation, and accounting mechanisms for storage that may be shared among multiple transient users. The **RFIO** and **DCAP** protocols were designed in the high-energy physics community to provide access to hierarchical mass storage devices such as Castor [5] and DCache [8].

Because Parrot must preserve POSIX semantics for the sake of the application, our foremost concern is the ability of each of these protocols to provide the necessary semantics. Performance is a secondary concern, although it is affected significantly by semantic issues. A summary of the semantics of each of these protocols is given in Figure 3.2.⁴

	name binding	discipline	dirs	metadata	symlinks	connections
posix	open/close	random	yes	direct	yes	-
chirp	open/close	random	yes	direct	yes	per client
ftp	get/put	sequential	varies	indirect	no	per file
nest	get/put	random	yes	indirect	yes	per client
rftio	open/close	random	yes	direct	no	per file/op
dcap	open/close	random	no	direct	no	per client

FIG. 3.2. *Protocol Compatibility with POSIX*

4. Errors and Boundary Conditions. Error handling has not been a pervasive problem in the design of traditional operating systems. As new models of file interaction have developed, attending error modes have been added to existing systems by expanding the software interface at every level. For example, the addition of distributed file systems to the Unix kernel created the new possibility of a stale file handle, represented by the *ESTALE* error. As this error mode was discovered at the very lowest layers of the kernel, the value was added to the device driver interface, the file system interface, the standard library, and expected to be handled directly by applications.

We have no such luxury in an interposition agent. Applications use the existing interface, and we have neither the desire nor the ability to change it. Sometimes, if we are lucky, we may re-use an error such as *ESTALE* for an analogous, if not identical purpose. Yet, the underlying device drivers generate errors ranging from the vague “file system error” to the microscopically precise “server’s certification authority is not trusted.” How should the unlimited space of errors in the lower layers be transformed into the fixed space of errors available to the application?⁵

For example, several device drivers have the necessary machinery to carry out all of a user’s possible requests, but provide vague errors when a supported operation fails. The FTP driver allows an application to read a file via the GET command. However, if the GET command fails, the only available information is the error code 550, which encompasses almost any sort of file system error including “no such file,” “access denied,” and “is a directory.” The POSIX interface does not permit a catch-all error value; it requires a specific reason. Which error code should be returned to the application?

One technique for dealing with this problem is to interview the service in order to narrow down the cause of the error, in a manner similar to that of an expert system. Suppose that we attempt to retrieve a file using an FTP GET operation. If the GET should fail, we may hypothesize that the named file is actually a directory. The hypothesis may be tested with a change directory (CWD) command. If that succeeds, the hypothesis is true, and we may return the precise error “not a file.” If that fails, we must propose another hypothesis and test it. Parrot performs a number of two- and three-step interviews in response to a variety of FTP errors.

The connection structure of a remote I/O protocol also has implications for semantics as well as performance. Chirp, NeST, and DCAP require one TCP connection between each client and server. FTP and RFIO require a new connection made for each file opened. In addition, RFIO requires a new connection for each operation performed on a non-open file. Because most file system operations are metadata queries, this can result in an

⁴Omitted: Details of the various protocols supported by Parrot.

⁵Omitted: Several more examples of error transformation.

extraordinary number of connections in a short amount of time. Ignoring the latency penalties of this activity, a large number of TCP connections can consume resources at clients, servers, and network devices such as address translators.⁶

5. Performance. We have deferred a discussion of performance until this point so that we may see the performance effects of semantic constraints. Although it is possible to write applications explicitly to use remote I/O protocols in the most efficient manner, Parrot must provide conservative and complete implementations of POSIX operations. For example, an application may only need to know the size of a file, but if it requests this information via *stat*, Parrot is obliged to fill the structure with everything it can, possibly at great cost.

The I/O services discussed here, with the exception of Chirp, are designed primarily for efficient high-volume data movement. This is demonstrated by Figure 5.1, which compares the throughput of the protocols at various block sizes. The throughput was measured by copying a 128 MB file into the remote storage device with the standard *cp* command equipped with Parrot and a varying default block size, as controlled through the *stat* emulation described above.

Of course, the absolute values are an artifact of our system, however, it can be seen that all of the protocols must be tuned for optimal performance. The exception is Chirp, which only reaches about one half of the available bandwidth. This is because of the strict RPC nature required for POSIX semantics; the Chirp server does not extract from the underlying filesystem any more data than necessary to supply the immediate read. Although it is technically feasible

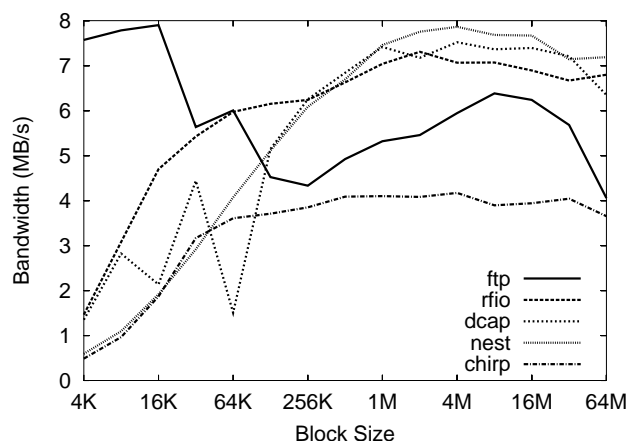


FIG. 5.1. Throughput of 128 MB File Copy

for the server to read ahead in anticipation of the next operation, such data pulled into the server’s address space might be invalidated by other actors on the file in the meantime and is thus semantically incorrect.

The hiccup in throughput of DCAP at a block size of 64KB is an unintended interaction with the default TCP buffer size of 64 KB. The developers of DCAP are aware of the artifact and recommend changing either the block size or the buffer size to avoid it. This is reasonable advice, given that all of the protocols require tuning of some kind.

Figure 5.2 benchmarks the latency of POSIX-equivalent operations in each I/O protocol. These measurements were obtained in a manner identical to that of Figure 2.1, with the indicated servers residing on the same system as in Figure 5.1. Notice that the latencies are measured in milliseconds, whereas Figure 2.1 gave microseconds.

proto	stat		open/close		read 8KB		write 8KB		bandwidth
chirp	.50±	.14 ms	.84±	.09	2.80±	.06	2.23±	.04	4.1 MB/s
ftp	.87±	.09 ms	2.82±	.26	(no random access)				7.9 MB/s
nest	2.51±	.05 ms	2.53±	.17	4.48±	.14	7.41±	.32	7.9 MB/s
rfio	13.41±	.28 ms	23.11±	1.29	3.32±	.14	2.85±	.18	7.3 MB/s
dcap	152.53±	16.68 ms	159.09±	16.68	3.01±	0.62	3.14±	.62	7.5 MB/s

FIG. 5.2. Performance of I/O Protocols On a Local-Area Network

We hasten to note that this comparison, in a certain sense, is not “fair.” These data servers provide vastly different services, so the performance differences demonstrate the cost of the service, not the cleverness of the implementation. For example, Chirp and FTP achieve low latencies because they are lightweight translation layers over an ordinary file system. NeST has somewhat higher latency because it provides the abstraction of a virtual file system, user namespace, access control lists, and a storage allocation system, all built on an existing filesystem. The cost is due to the necessary metadata log that records all such activity that cannot be stored directly in the underlying file system. Both RFIO and DCAP are designed to interact with mass storage

⁶Omitted: A discussion of the interface between Parrot and batch systems.

dist.	proto	copy	list	scan	make	delete
local	local	.15± .02 <i>sec</i>	.09± .20	.08± .02	65.38±3.47	.86± .18 <i>sec</i>
local	chirp	1.22± .03 <i>sec</i>	.34± .02	.40± .01	81.02±1.46	.79± .01 <i>sec</i>
lan	chirp	6.16± .22 <i>sec</i>	.57± .30	1.32± .03	144.00±1.35	1.26± .02 <i>sec</i>
lan	chirp	10.67± .90 <i>sec</i>	.53± .07	4.72± .32	95.05±2.33	1.24± .03 <i>sec</i>
lan	ftp	34.88±1.72 <i>sec</i>	1.47± .02	17.78±1.14	122.54±3.14	2.95± .15 <i>sec</i>
lan	nest	52.35±4.18 <i>sec</i>	12.92±4.87	28.14±4.52	307.19±3.26	31.73±4.37 <i>sec</i>
lan	rfio	<i>(overwhelmed by repeated connections)</i>				
lan	dcap	<i>(does not support directories without nfs)</i>				

FIG. 5.3. Performance of the Andrew-Like Benchmark

systems; single operations may result in gigabytes of activity within a disk cache, possibly moving files to or from tape. In that context, low latency is not a concern.

That said, several things may be observed from this table. Although FTP has benefitted from years of optimizations, the cost of a *stat* is greater than that of Chirp because of the need for multiple round trips to fill in the necessary details. The additional latency of *open/close* is due to the multiple round trips to name and establish a new TCP connection. Both RFIO and DCAP have higher latencies for single byte reads and writes than for 8KB reads and writes. This is due to buffering which delays small operations in anticipation of further data. Most importantly, all of these remote operations exceed the latency of the debugger trap itself by several orders of magnitude. Thus, we are comfortable with the previous decision to sacrifice performance in favor of reliability in the interposition technique.

We conclude with a macrobenchmark similar to the Andrew benchmark. [11] This Andrew-like benchmark consists of a series of operations on the Parrot source tree, which consists of 13 directories and 296 files totaling 955 KB. To prepare, the source tree is moved to the remote device. In the **copy** stage, the tree is duplicated on the remote device. In the **list** stage, a detailed list (`ls -lR`) of the tree is made. In the **scan** stage, all files in the tree are searched (`grep`) for a text string. In the **make** stage, the software is built. From an I/O perspective, this involves a sequential read of every source file, a sequential write of every object file, and a series of random reads and writes to create the executables. In the **delete** stage, the tree is deleted.

Figure 5.3 compares the performance of the Andrew-like benchmark in a variety of configurations. In the three cases above the horizontal rule, we measure the cost of each layer of software added: first with Parrot only, then with a Chirp server on the same host, then with a Chirp server across the local area network. Not surprisingly, the I/O cost of separating computation from storage is high. Copying data is much slower over the network, although the slowdown in the make stage is quite acceptable if we intend to increase throughput via remote parallelization.

In the two cases adjacent to the rule, the only change is the enabling of caching. As might be expected, the cost of unnecessary duplication causes an increase in copying the source tree, although the difference is easily made up in the make stage, where the cache eliminates the multiple random I/O necessary to link executables. The list and delete stages only involve directory structure and metadata access and are thus not affected by the cache.

In the five cases below the horizontal rule, we explore the use of various protocols to run the benchmark. In all of these cases, caching is enabled in order to eliminate the cost of random access as discussed. The DCAP protocol is semantically unable to run the benchmark, as it does not provide the necessary access to directories. The RFIO protocol is semantically able to run the benchmark, but the high frequency of filesystem operations results in a large number of TCP connections, which quickly exhausts networking resources at both the client and the server, thus preventing the benchmark from running. Chirp, FTP, and NeST are all able to complete the benchmark. The NeST results have a high variance, due to delays incurred while the metadata log is periodically compressed. The difference in performance between Chirp, FTP, and NeST is primarily attributable to the cost of metadata lookups. All the stages make heavy use of *stat*; the multiple round trips necessary to implement this completely for FTP and NeST have a striking cumulative effect.

6. Conclusions. Interposition agents provide a stable platform for bringing old applications into new environments. We have outlined the difficulties that we have encountered as well as the solutions we have constructed in the course of building and deploying several types of agents within the Condor project. As we have shown, the Linux debugger trap has several limitations, but can still be put to good use. As interest grows

in the use of virtual machines in distributed systems [26] the need for powerful but low overhead methods of interposition grows. The appropriate interface for this task is still an open research topic.

The notion of virtualizing or multiplexing an existing interface is a common technique [14, 7], but the plague of errors and other boundary conditions seems to be suffered silently by practitioners. Such problems are rarely publicized, however, we are aware of two excellent exceptions. C. Metz [16] describes how the Berkeley sockets interface is surprisingly hard to multiplex. T. Garfinkel [10] describes the subtle semantic problems of sandboxing untrusted applications.

For more information: <http://www.cs.wisc.edu/~thain/research/parrot>

7. Acknowledgments. We thank John Bent and Sander Klous for their help deploying and debugging Parrot. Victor Zandy wrote the mechanism for binary rewriting. Alain Roy gave thoughtful comments on early drafts of this paper.

REFERENCES

- [1] M. ACCETTA, R. BARON, W. BOLOSKY, D. GOLUB, R. RASHID, A. TEVANI, AND M. YOUNG, *Mach: A new kernel foundation for Unix development*, in Proceedings of the USENIX Summer Technical Conference, Atlanta, GA, 1986.
- [2] A. ALEXANDROV, M. IBEL, K. SCHAUER, AND C. SCHEIMAN, *UFO: A personal global file system based on user-level extensions to the operating system*, ACM Transactions on Computer Systems, (1998), pp. 207–233.
- [3] W. ALLCOCK, A. CHERVENAK, I. FOSTER, C. KESSELMAN, AND S. TUECKE, *Protocols and services for distributed data-intensive science*, in Proceedings of Advanced Computing and Analysis Techniques in Physics Research, 2000, pp. 161–163.
- [4] D. ANDERSON, J. CHASE, AND A. VAHDAT, *Interposed request routing for scalable network storage*, in Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, 2000.
- [5] O. BARRING, J. BAUD, AND J. DURAND, *CASTOR project status*, in Proceedings of Computing in High Energy Physics, Padua, Italy, 2000.
- [6] J. BENT, V. VENKATARAMANI, N. LEROY, A. ROY, J. STANLEY, A. ARPACI-DUSSEAU, R. ARPACI-DUSSEAU, AND M. LIVNY, *Flexibility, manageability, and performance in a grid storage appliance*, in Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing, Edinburgh, Scotland, July 2002.
- [7] D. CHERITON, *UIO: A uniform I/O system interface for distributed systems*, ACM Transactions on Computer Systems, 5 (1987), pp. 12–46.
- [8] M. ERNST, P. FUHRMANN, M. GASTHUBER, T. MKRTCHYAN, AND C. WALDMAN, *dCache, a distributed storage data caching system*, in Proceedings of Computing in High Energy Physics, Beijing, China, 2001.
- [9] I. FOSTER, C. KESSELMAN, G. TSUDIK, AND S. TUECKE, *A security architecture for computational grids*, in Proceedings of the 5th ACM Conference on Computer and Communications Security Conference, 1998, pp. 83–92.
- [10] T. GARFINKEL, *Traps and pitfalls: Practical problems in in system call interposition based security tools*, in Proceedings of the Network and Distributed Systems Security Symposium, February 2003.
- [11] J. HOWARD, M. KAZAR, S. MENEES, D. NICHOLS, M. SATYANARAYANAN, R. SIDEBOTHAM, AND M. WEST, *Scale and performance in a distributed file system*, ACM Transactions on Computer Systems, 6 (1988), pp. 51–81.
- [12] G. HUNT AND D. BRUBACHER, *Detours: Binary interception of Win32 functions*, Tech. Report MSR-TR-98-33, Microsoft Research, February 1999.
- [13] M. JONES, *Interposition agents: Transparently interposing user code at the system interface*, in Proceedings of the 14th ACM Symposium on Operating Systems Principles, 1993.
- [14] S. KLEIMAN, *Vnodes: An architecture for multiple file system types in Sun Unix*, in Proceedings of the USENIX Technical Conference, 1986, pp. 151–163.
- [15] M. LEECH, M. GANIS, Y. LEE, R. KURIS, D. KOBLAS, AND L. JONES, *SOCKS protocol version 5*. Internet Engineering Task Force, Request for Comments 1928, March 1996.
- [16] C. METZ, *Protocol independence using the sockets API*, in Proceedings of the USENIX Technical Conference, June 2002.
- [17] B. MILLER, M. CALLAGHAN, J. CARGILLE, J. HOLLINGSWORTH, R. B. IRVIN, K. KARAVANIC, K. KUNCHITHAPADAM, AND T. NEWHALL, *The Paradyn parallel performance measurement tools*, IEEE Computer, 28 (1995), pp. 37–46.
- [18] B. MILLER, A. CORTES, M. A. SENAR, AND M. LIVNY, *The tool daemon protocol (TDP)*, in Proceedings of Supercomputing, Phoenix, AZ, November 2003.
- [19] C. SMALL AND M. SELTZER, *A comparison of OS extension technologies*, in Proceedings of the USENIX Technical Conference, 1996, pp. 41–54.
- [20] M. SOLOMON AND M. LITZKOW, *Supporting checkpointing and process migration outside the Unix kernel*, in Proceedings of the USENIX Winter Technical Conference, 1992.
- [21] D. THAIN AND M. LIVNY, *Multiple bypass: Interposition agents for distributed computing*, Journal of Cluster Computing, 4 (2001), pp. 39–47.
- [22] ———, *Error scope on a computational grid*, in Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing, July 2002.
- [23] ———, *Parrot: Transparent user-level middleware for data-intensive computing*, in Proceedings of the Workshop on Adaptive Grid Middleware, September 2003.
- [24] ———, *Parrot: Transparent user-level middleware for data-intensive computing*, Tech. Report 1493, Computer Sciences Department, University of Wisconsin, December 2003.
- [25] K.-P. VO, *The discipline and method architecture for reusable libraries*, Software: Practice and Experience, 30 (2000), pp. 107–128.

- [26] A. WHITAKER, M. SHAW, AND S. D. GRIBBLE, *Scale and performance in the Denali isolation kernel*, in Proceedings of the Fifth Symposium on Operating System Design and Implementation, Boston, MA, December 2002.
- [27] B. WHITE, A. GRIMSHAW, AND A. NGUYEN-TUONG, *Grid-Based File Access: The Legion I/O Model*, in Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing, August 2000.
- [28] V. ZANDY, B. MILLER, AND M. LIVNY, *Process hijacking*, in Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing, 1999.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 14, 2003.

Accepted: September 1, 2003.



SATIN: SIMPLE AND EFFICIENT JAVA-BASED GRID PROGRAMMING

ROB V. VAN NIEUWPOORT, JASON MAASSEN, THILO KIELMANN, HENRI E. BAL*

Abstract. Grid programming environments need to be both *portable* and *efficient* to exploit the computational power of dynamically available resources. In previous work, we have presented the divide-and-conquer based Satin model for parallel computing on clustered wide-area systems. In this paper, we present the Satin implementation on top of our new Ibis platform which combines Java's *write once, run everywhere* with efficient communication between JVMs. We evaluate Satin/Ibis on the testbed of the EU-funded GridLab project, showing that Satin's load-balancing algorithm automatically adapts both to heterogeneous processor speeds and varying network performance, resulting in efficient utilization of the computing resources. Our results show that when the wide-area links suffer from congestion, Satin's load-balancing algorithm can still achieve around 80% efficiency, while an algorithm that is not grid aware drops to 26% or less.

Key words. Satin, Ibis, divide-and-conquer, load balancing, distributed supercomputing.

1. Introduction. In computational grids, applications need to simultaneously tap the computational power of multiple, dynamically available sites. The crux of designing grid programming environments stems exactly from the dynamic availability of compute cycles: grid programming environments need to be both *portable* to run on as many sites as possible, and they need to be *flexible* to cope with different network protocols and dynamically changing groups of heterogeneous compute nodes.

Existing programming environments are either portable and flexible (Jini, Java RMI), or they are highly efficient (MPI). The Global Grid Forum also has investigated possible grid programming models [19]. Recently, *GridRPC* has been proposed as a grid programming model [30]. GridRPC allows writing grid applications based on the manager/worker paradigm.

Unlike manager/worker programs, divide-and-conquer algorithms operate by recursively dividing a problem into smaller subproblems. This recursive subdivision goes on until the remaining subproblem becomes trivial to solve. After solving subproblems, their results are recursively recombined until the final solution is assembled. By allowing subproblems to be divided recursively, the class of divide-and-conquer algorithms subsumes the manager/worker algorithms, thus enlarging the set of possible grid applications.

Of course, there are many kinds of applications that do not lend themselves well to a divide-and-conquer algorithm. However, we (and others) believe the class of divide-and-conquer algorithms to be sufficiently large to justify its deployment for hierarchical wide-area systems. Computations that use the divide-and-conquer model include geometry procedures, sorting methods, search algorithms, data classification codes, n-body simulations and data-parallel numerical programs [33].

Divide-and-conquer applications may be parallelized by letting different processors solve different subproblems. These subproblems are often called *jobs* in this context. Generated jobs are transferred between processors to balance the load in the computation. The divide-and-conquer model lends itself well to hierarchically-structured systems because tasks are created by recursive subdivision. This leads to a task graph that is hierarchically structured, and which can be executed with excellent communication locality, especially on hierarchical platforms.

In previous work [26], we presented our *Satin* system for divide-and-conquer programming on grid platforms. Satin implements a very efficient load balancing algorithm for clustered, wide-area platforms. So far, we could only evaluate Satin based on simulations in which all jobs have been executed on one single, homogeneous cluster. In this work, we evaluate Satin on a real grid testbed [2], consisting of various heterogeneous systems, connected by the Internet.

In Section 2, we briefly present Satin's programming model and some simulator-based results that indicate the suitability of Satin as a grid programming environment. In Section 3, we present Ibis, our new Java-based grid programming platform that combines Java's "run everywhere" paradigm with highly efficient yet flexible communication mechanisms. In Section 4, we evaluate the performance of Satin on top of Ibis in the GridLab testbed, spanning several sites in Europe. Section 5 discusses related work, and in Section 6 we draw conclusions.

*Dept. of Computer Science, Vrije Universiteit, Amsterdam, The Netherlands, {rob,jason,kielmann,bal}@cs.vu.nl
<http://www.cs.vu.nl/ibis>

2. Divide-and Conquer in Satin. Satin’s programming model is an extension of the single-threaded Java model. To achieve parallel execution, Satin programs do not have to use Java’s threads or Remote Method Invocations (RMI). Instead, they use much simpler divide-and-conquer primitives. Satin does allow the combination of its divide-and-conquer primitives with Java threads and RMIs. Additionally, Satin provides shared objects via RepMI. In this paper, however, we focus on pure divide-and-conquer programs.

```
interface FibInter extends satin.Spawnable {
    public long fib(long n);
}

class Fib extends satin.SatinObject
    implements FibInter {
    public long fib(long n) {
        if(n < 2) return n;

        long x = fib(n-1); // spawned
        long y = fib(n-2); // spawned
        sync();

        return x + y;
    }

    public static void main(String[] args) {
        Fib f = new Fib();
        long res = f.fib(10);
        f.sync();
        System.out.println("Fib_10_=_ " + res);
    }
}
```

FIG. 2.1. Fib: an example divide-and-conquer program in Satin.

Satin expresses divide-and-conquer parallelism entirely in the Java language itself, without requiring any new language constructs. Satin uses so-called *marker interfaces* to indicate that certain method invocations need to be considered for potentially parallel (so called *spawned*) execution, rather than being executed synchronously like normal methods. Furthermore, a mechanism is needed to synchronize with (wait for the results of) spawned method invocations. With Satin, this can be expressed using a special interface, `satin.Spawnable`, and the class `satin.SatinObject`. This is shown in Fig. 2.1, using the example of a class `Fib` for computing the Fibonacci numbers. First, an interface `FibInter` is implemented which extends `satin.Spawnable`. All methods defined in this interface (here `fib`) are marked to be spawned rather than executed normally. Second, the class `Fib` extends `satin.SatinObject` and implements `FibInter`. From `satin.SatinObject` it inherits the `sync` method, from `FibInter` the spawned `fib` method. Finally, the invoking method (in this case `main`) simply calls `Fib` and uses `sync` to wait for the result of the parallel computation.

Satin’s byte code rewriter generates the necessary code. Conceptually, a new thread is started for running a spawned method upon invocation. Satin’s implementation, however, eliminates thread creation altogether. A spawned method invocation is put into a local work queue. From the queue, the method might be transferred to a different CPU where it may run concurrently with the method that executed the spawned method. The `sync` method waits until all spawned calls in the current method invocation are finished; the return values of spawned method invocations are undefined until a `sync` is reached.

Spawned method invocations are distributed across the processors of a parallel Satin program by work stealing from the work queues mentioned above. In [26], we presented a new work stealing algorithm, *Cluster-aware Random Stealing* (CRS), specifically designed for cluster-based, wide-area (grid computing) systems. CRS is based on the traditional Random Stealing (RS) algorithm that has been proven to be optimal for homogeneous (single cluster) systems [8]. We briefly describe both algorithms in turn.

2.1. Random Stealing (RS). RS attempts to steal a job from a randomly selected peer when a processor finds its own work queue empty, repeating steal attempts until it succeeds [8, 33]. This approach minimizes communication overhead at the expense of idle time. No communication is performed until a node becomes idle, but then it has to wait for a new job to arrive. On a single-cluster system, RS is the best performing

load-balancing algorithm. On wide-area systems, however, this is not the case. With C clusters, on average $(C - 1)/C \times 100\%$ of all steal requests will go to nodes in remote clusters, causing significant wide-area communication overheads.

2.2. Cluster-aware Random Stealing (CRS). In CRS, each node can directly steal jobs from nodes in remote clusters, but at most one job at a time. Whenever a node becomes idle, it first attempts to steal from a node in a remote cluster. This wide-area steal request is sent asynchronously: Instead of waiting for the result, the thief simply sets a flag and performs additional, synchronous steal requests to randomly selected nodes within its own cluster, until it finds a new job. As long as the flag is set, only local stealing will be performed. The handler routine for the wide-area reply simply resets the flag and, if the request was successful, puts the new job into the work queue. CRS combines the advantages of RS inside a cluster with a very limited amount of asynchronous wide-area communication. Below, we will show that CRS performs almost as good as with a single, large cluster, even in extreme wide-area network settings.

2.3. Simulator-based comparison of RS and CRS. A detailed description of Satin’s wide-area work stealing algorithm can be found in [26]. We have extracted the comparison of RS and CRS from that work into Table 2.1. The run times shown in this table are for parallel runs with 64 CPUs each, either with a single cluster of 64 CPUs, or with 4 clusters of 16 CPUs each.

The wide-area network between the virtual clusters has been simulated with our Panda WAN simulator [17]. We simulated all combinations of 20 ms and 200 ms roundtrip latency with bandwidth capacities of 100 KByte/s and 1000 KByte/s. The tests had been performed on the predecessor hardware to our current DAS-2 cluster. DAS consists of 200 MHz Pentium Pro’s with a Myrinet network, running the Manta parallel Java system [23].

TABLE 2.1
Performance of RS and CRS with different simulated wide-area links (times in seconds).

application	single cluster		20 ms 1000 KByte/s		20 ms 100 KByte/s		200 ms 1000 KByte/s		200 ms 100 KByte/s	
	time	eff.	time	eff.	time	eff.	time	eff.	time	eff.
adaptive integration										
RS	71.8	99.6%	78.0	91.8%	79.5	90.1%	109.3	65.5%	112.3	63.7%
CRS	71.8	99.7%	71.6	99.9%	71.7	99.8%	73.4	97.5%	73.2	97.7%
N-queens										
RS	157.6	92.5%	160.9	90.6%	168.2	86.6%	184.3	79.1%	197.4	73.8%
CRS	156.3	93.2%	158.1	92.2%	156.1	93.3%	158.4	92.0%	158.1	92.2%
TSP										
RS	101.6	90.4%	105.3	87.2%	105.4	87.1%	130.6	70.3%	129.7	70.8%
CRS	100.7	91.2%	103.6	88.7%	101.1	90.8%	105.0	87.5%	107.5	85.4%
ray tracer										
RS	147.8	94.2%	152.1	91.5%	171.6	81.1%	175.8	79.2%	182.6	76.2%
CRS	147.2	94.5%	145.0	95.9%	152.6	91.2%	146.5	95.0%	149.3	93.2%

In Table 2.1 we compare RS and CRS using four parallel applications, with network conditions degrading from the left (single cluster) to the right (high latency, low bandwidth). For each case, we present the parallel run time and the corresponding efficiency (labeled “eff.” in the table). With t_s being the sequential run time for the application, with the Satin operations excluded, (not shown) and t_p the parallel run time as shown in the table, and $N = 64$ being the number of CPUs, we compute the efficiency as follows:

$$efficiency = \frac{t_s}{t_p \cdot N} * 100\%$$

Adaptive integration numerically integrates a function over a given interval. It sends very short messages and has also very fine grained jobs. This combination makes RS sensitive to high latency, in which case efficiency drops to about 65%. CRS, however, successfully hides the high round trip times and achieves efficiencies of more than 97% in all cases.

N Queens solves the problem of placing n queens on a $n \times n$ chess board. It sends medium-size messages and has a very irregular task tree. With efficiency of only 74%, RS again suffers from high round trip times as it can not quickly compensate load imbalance due to the irregular task tree. CRS, however, sustains efficiencies of 92%.

TSP solves the problem of finding the shortest path between n cities. By passing the distance table as parameter, it has a somewhat higher parallelization overhead, resulting in slightly lower efficiencies, even with a single cluster. In the wide-area cases, these longer parameter messages contribute to higher round trip times when stealing jobs from remote clusters. Consequently, RS suffers more from slower networks (efficiency $> 70\%$) than CRS which sustains efficiencies of 85%.

Ray Tracer renders a modeled scene to a raster image. It divides a screen down to jobs of single pixels. Due to the nature of ray tracing, individual pixels have very irregular rendering times. The application sends long result messages containing image fractions, making it sensitive to the available bandwidth. This sensitivity is reflected in the efficiency of RS, going down to 76%, whereas CRS hides most WAN communication overhead and sustains efficiencies of 91%.

To summarize, our simulator-based experiments show the superiority of CRS to RS in case of multiple clusters, connected by wide-area networks. This superiority is independent of the properties of the applications, as we have shown with both regular and irregular task graphs as well as short and long parameter and result message sizes. In all investigated cases, the efficiency of CRS never dropped below 85%.

Although we were able to identify the individual effects of wide-area latency and bandwidth, these results are limited to homogeneous Intel/Linux clusters (due to the Manta compiler). Furthermore, we only tested clusters of identical size. Finally, the wide area network has been simulated and thus been without possibly disturbing third-party traffic.

An evaluation on a real grid testbed, with heterogeneous CPUs, JVMs, and networks, becomes necessary to prove the suitability of Satin as a grid programming platform. In the following, we first present Ibis, our new *run everywhere* Java environment for grid computing. Then we evaluate Satin on top of Ibis on the testbed of the EU GridLab project.

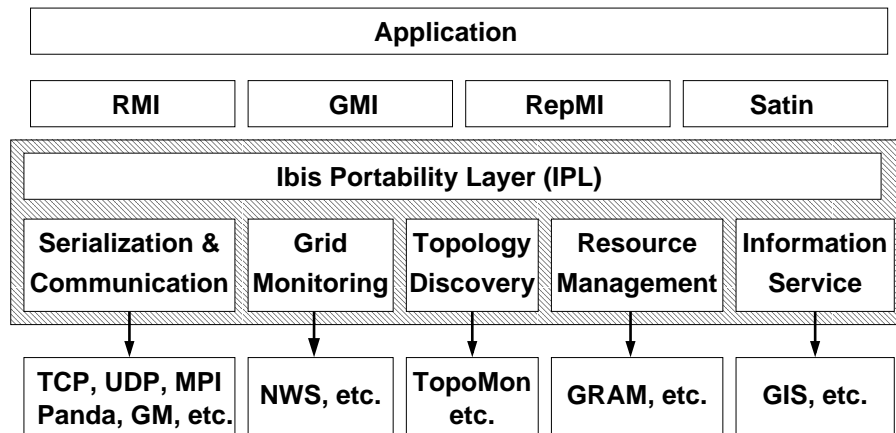


FIG. 3.1. Design of Ibis. The various modules can be loaded dynamically, using run time class loading.

3. Ibis, flexible and efficient Java-based Grid programming. The Satin runtime system used for this paper is implemented on top of Ibis [31]. In this section we will briefly explain the Ibis philosophy and design. The global structure of the Ibis system is shown in Figure 3.1. A central part of the system is the Ibis Portability Layer (IPL) which consists of a number of well-defined interfaces. The IPL can have different implementations, that can be selected and loaded into the application *at run time*. The IPL defines serialization and communication, but also typical grid services such as topology discovery and monitoring. Although it is possible to use the IPL directly from an application, Ibis also provides more high-level programming models. Currently, we have implemented four. Ibis RMI [31] provides Remote Method Invocation, using the same interface as Sun RMI, but with a more efficient wire protocol. GMI [21] provides MPI-like collective operations, cleanly integrated into Java's object model. RepMI [22] extends Java with replicated objects. In this paper, we focus on the fourth programming model that Ibis implements, Satin.

3.1. Ibis Goals. A key problem in making Java suitable for grid programming is how to design a system that obtains high communication performance while still adhering to Java's "write once, run everywhere" model. Current Java implementations are heavily biased to either portability or performance, and fail in the other

aspect. (The recently added *java.nio* package will hopefully at least partially address this problem). The Ibis strategy to achieve both goals simultaneously is to develop reasonably efficient solutions using standard techniques that work “everywhere”, supplemented with highly optimized but non-standard solutions for increased performance in special cases. We apply this strategy to both computation and communication. Ibis is designed to use any standard JVM, but if a native, optimizing compiler (e.g., Manta [23]) is available for a target machine, Ibis can use it instead. Likewise, Ibis can use standard communication protocols, e.g., TCP/IP or UDP, as provided by the JVM, but it can also plug in an optimized low-level protocol for a high-speed interconnect, like GM or MPI, if available. The challenges for Ibis are:

1. how to make the system flexible enough to run *seamlessly* on a variety of different communication hardware and protocols;
2. how to make the standard, 100% pure Java case efficient enough to be useful for grid computing;
3. study which additional optimizations can be done to improve performance further in special (high-performance) cases.

With Ibis, grid applications can run simultaneously on a variety of different machines, using optimized software where possible (e.g., a native compiler, the GM Myrinet protocol, or MPI), and using standard software (e.g., TCP) when necessary. Interoperability is achieved by using the TCP protocol between multiple Ibis implementations that use different protocols (like GM or MPI) locally. This way, all machines can be used in one single computation. Below, we discuss the three aforementioned issues in more detail.

3.2. Flexibility. The key characteristic of Ibis is its extreme flexibility, which is required to support grid applications. A major design goal is the ability to seamlessly plug in different communication substrates without changing the user code. For this purpose, the Ibis design uses the IPL. A software layer on top of the IPL can negotiate with Ibis instantiations through the well-defined IPL interface, to select and load the modules it needs. This flexibility is implemented using Java’s dynamic class-loading mechanism.

Many message passing libraries such as MPI and GM guarantee reliable message delivery and FIFO message ordering. When applications do not require these properties, a different message passing library might be used to avoid the overhead that comes with reliability and message ordering. The IPL supports both reliable and unreliable communication, ordered and unordered messages, implicit and explicit receipt, using a single, simple interface. Using user-definable properties (key-value pairs), applications can create exactly the communication channels they need, without unnecessary overhead.

3.3. Optimizing the Common Case. To obtain acceptable communication performance, Ibis implements several optimizations. Most importantly, the overhead of serialization and reflection is avoided by compile-time generation of special methods (in byte code) for each object type. These methods can be used to convert objects to bytes (and vice versa), and to create new objects on the receiving side, without using expensive reflection mechanisms. This way, the overhead of serialization is reduced dramatically.

Furthermore, our communication implementations use an optimized wire protocol. The Sun RMI protocol, for example, resends type information for each RMI. Our implementation caches this type information per connection. Using this optimization, our protocol sends less data over the wire, but more importantly, saves processing time for encoding and decoding the type information.

3.4. Optimizing Special Cases. In many cases, the target machine may have additional facilities that allow faster computation or communication, which are difficult to achieve with standard Java techniques. One example we investigated in previous work [23] is using a native, optimizing compiler instead of a JVM. This compiler (Manta), or any other high performance Java implementation, can simply be used by Ibis. The most important special case for communication is the presence of a high-speed local interconnect. Usually, specialized user-level network software is required for such interconnects, instead of standard protocols (TCP, UDP) that use the OS kernel. Ibis therefore was designed to allow other protocols to be plugged in. So, lower-level communication may be based, for example, on a locally-optimized MPI library. The IPL is designed in such a way that it is possible to exploit efficient hardware multicast, when available.

Another important feature of the IPL is that it allows a zero-copy implementation. Implementing zero-copy (or single-copy) communication in Java is a non-trivial task, but it is essential to make Java competitive with systems like MPI for which zero-copy implementations already exist. The zero-copy Ibis implementation is described in more detail in [31]. On fast networks like Myrinet, the throughput of Ibis RMI can be as much as 9 times higher than previous, already optimized RMI implementations such as KaRMI [28].

4. Satin on the GridLab testbed. In this section, we will present a case study to analyze the performance that Satin/Ibis achieves in a *real* grid environment. We ran the ray tracer application introduced in Section 2.3 on the European GridLab [2] testbed. More precisely, we were using a characteristic subset of the machines on this testbed that was available for our measurements at the time the study was performed. Because simultaneously starting and running a parallel application on multiple clusters still is a tedious and time-consuming task, we had to restrict ourselves to a single test application. We have chosen the ray tracer for our tests as it is sending the most data of all our applications, making it very sensitive to network issues. The ray tracer is written in pure Java and generates a high resolution image (4096×4096 , with 24-bit color). It takes approximately 10 minutes to solve this problem on our testbed.

This is an interesting experiment for several reasons. Firstly, we use the Ibis implementation on top of TCP for the measurements in this section. This means that the numbers shown below were measured using a 100% Java implementation. Therefore, they are interesting, giving a clear indication of the performance level that can be achieved in Java with a “run everywhere” implementation, without using any native code.

Secondly, the testbed contains machines with several different architectures; Intel, SPARC, MIPS, and Alpha processors are used. Some machines are 32 bit, while others are 64 bit. Also, different operating systems and JVMs are in use. Therefore, this experiment is a good method to investigate whether Java’s “write once, run everywhere” feature really works in practice. The assumption that this feature successfully hides the complexity of the different underlying architectures and operating systems, was the most important reason for investigating the Java-centric solutions presented in this paper. It is thus important to verify the validity of this claim.



FIG. 4.1. Locations of the GridLab testbed sites used for the experiments.

Thirdly, the machines are connected by the Internet. The links show typical wide-area behavior, as the physical distance between the sites is large. For instance, the distance from Amsterdam to Lecce is roughly 2000 kilometers (about 1250 miles). Figure 4.1 shows a map of Europe, annotated with the machine locations. This gives an idea of the distances between the sites. We use this experiment to verify Satin's load-balancing algorithms in practice, with *real* non-dedicated wide-area links. We have run the ray tracer both with the standard random stealing algorithm (RS) and with the new cluster-aware algorithm (CRS) as introduced above. For practical reasons, we had to use relatively small clusters for the measurements in this section. The simulation results in Section 2.3 show that the performance of CRS increases when larger clusters are used, because there is more opportunity to balance the load inside a cluster during wide-area communication.

TABLE 4.1
Machines on the GridLab testbed.

location	architecture	Operating System	JIT	nodes	CPUs / node	total CPUs
Vrije Universiteit <i>Amsterdam</i> <i>The Netherlands</i>	Intel Pentium-III 1 GHz	Red Hat Linux kernel 2.4.18	IBM 1.4.0	8	1	8
Vrije Universiteit <i>Amsterdam</i> <i>The Netherlands</i>	Sun Fire 280R UltraSPARC-III 750 MHz 64 bit	Sun Solaris 8	SUN HotSpot 1.4.2	1	2	2
ISUFI/High Perf. Computing Center <i>Lecce, Italy</i>	Compaq Alpha 667 MHz 64 bit	Compaq Tru64 UNIX V5.1A	HP 1.4.0 based on HotSpot	1	4	4
Cardiff University <i>Cardiff, Wales, UK</i>	Intel Pentium-III 1 GHz	Red Hat Linux 7.1 kernel 2.4.2	SUN HotSpot 1.4.1	1	2	2
Masaryk University, <i>Brno, Czech Republic</i>	Intel Xeon 2.4 GHz	Debian Linux kernel 2.4.20	IBM 1.4.0	4	2	8
Konrad-Zuse-Zentrum für Informationstechnik <i>Berlin, Germany</i>	SGI Origin 3000 MIPS R14000 500 MHz	IRIX 6.5	SGI 1.4.1-EA based on HotSpot	1	16	16

Some information about the machines we used is shown in Table 4.1. To run the application, we used whichever Java JIT (Just-In-Time compiler) that was pre-installed on each particular system whenever possible, because this is what most users would probably do in practice.

TABLE 4.2
Round-trip wide-area latency (in milliseconds) and achievable bandwidth (in KByte/s) between the GridLab sites.

source	daytime						nighttime					
	to A'dam DAS-2	to A'dam Sun	to Lecce	to Cardiff	to Brno	to Berlin	to A'dam DAS-2	to A'dam Sun	to Lecce	to Cardiff	to Brno	to Berlin
latency from												
A'dam DAS-2	—	1	204	16	20	42	—	1	65	15	20	18
A'dam Sun	1	—	204	15	19	43	1	—	62	14	19	17
Lecce	198	195	—	210	204	178	63	66	—	60	66	64
Cardiff	9	9	198	—	28	26	9	9	51	—	27	21
Brno	20	20	188	33	—	22	20	19	64	33	—	22
Berlin	18	17	185	31	22	—	18	17	59	30	22	—
bandwidth from												
A'dam DAS-2	—	11338	42	750	3923	2578	—	11442	40	747	4115	2578
A'dam Sun	11511	—	22	696	2745	2611	11548	—	46	701	3040	2626
Lecce	73	425	—	44	43	75	77	803	—	94	110	82
Cardiff	842	791	29	—	767	825	861	818	37	—	817	851
Brno	3186	2709	26	588	—	2023	3167	2705	37	612	—	2025
Berlin	2555	2633	9	533	2097	—	2611	2659	9	562	2111	—

Because the sites are connected via the Internet, we have no influence on the amount of traffic that flows over the links. To reduce the influence of Internet traffic on the measurements, we also performed measurements after midnight (CET). However, in practice there still is some variability in the link speeds. We measured the latency of the wide-area links by running *ping* 50 times, while the achievable bandwidth is measured with *netperf* [25], using 32 KByte packets. The measured latencies and bandwidths are shown in Table 4.2. All sites had difficulties from time to time while sending traffic to Lecce, Italy. For instance, from Amsterdam to Lecce, we measured latencies from 44 milliseconds up to 3.5 seconds. Also, we experienced packet loss with this link: up to 23% of the packets were dropped along the way. We also performed the same measurement during daytime, to investigate how regular Internet traffic influences the application performance. The measurements show that

there can be more than a factor of two difference in link speeds during daytime and nighttime, especially the links from and to Lecce show a large variability. It is also interesting to see that the link performance from Lecce to the two sites in Amsterdam is different. We verified this with *traceroute*, and found that the traffic is indeed routed differently as the two machines use different network numbers despite being located within the same building.

TABLE 4.3
Problems encountered in a real grid environment, and their solutions.

problem	solution
firewalls	bind all sockets to ports in the open range
buggy JITs	upgrade to Java 1.4 JITs
multi-homes machines	use a single, externally valid IP address

Ibis, Satin and the ray tracer application were all compiled with the standard Java compiler *javac* on the DAS-2 machine in Amsterdam, and then just copied to the other GridLab sites, without recompiling or reconfiguring anything. On most sites, this works flawlessly. However, we did run into several practical problems. A summary is given in Table 4.3. Some of the GridLab sites have firewalls installed, which block Satin's traffic when no special measures are taken. Most sites in our testbed have some open port range, which means that traffic to ports within this range can pass through. The solution we use to avoid being blocked by firewalls is straightforward: all sockets used for communication in Ibis are bound to a port within the (site-specific) open port range. We are working on a more general solution that multiplexes all traffic over a single port. Another solution is to multiplex all traffic over a (Globus) ssh connection, as is done by Kaneda et al. [16], or using a mechanism like SOCKS [20].

Another problem we encountered was that the JITs installed on some sites contained bugs. Especially the combination of threads and sockets presented some difficulties. There seems to be a bug in Sun's 1.3 JIT (HotSpot) related to threads and socket communication. In some circumstances, a blocking operation on a socket would block the whole application instead of just the thread that does the operation. The solution for this problem was to upgrade to a Java 1.4 JIT, where the problem is solved.

Finally, some machines in the testbed are multi-homed: they have multiple IP addresses. The original Ibis implementation on TCP got confused by this, because the *InetAddress.getLocalHost* method can return an IP address in a private range, or an address for an interface that is not accessible from the outside. Our current solution is to manually specify which IP address has to be used when multiple choices are available. All machines in the testbed have a Globus [10] installation, so we used GSI-SSH (Globus Security Infrastructure Secure Shell) [11] to login to the GridLab sites. We had to start the application by hand, as not all sites have a job manager installed. When a job manager is present, Globus can be used to start the application automatically.

As shown in Table 4.1, we used 40 processors in total, using 6 machines located at 5 sites all over Europe, with 4 different processor architectures. After solving the aforementioned practical problems, Satin on the TCP Ibis implementation ran on all sites, in pure Java, without having to recompile anything.

TABLE 4.4
Relative speeds of the machine and JVM combinations in the testbed.

site	architecture	run time (s)	relative node speed	relative total speed of cluster	% of total system
A'dam DAS-2	1 GHz Intel Pentium-III	233.1	1.000	8.000	32.4
A'dam Sun	750 MHz UltraSPARC-III	445.2	0.523	1.046	4.2
Lecce	667 MHz Compaq Alpha	512.7	0.454	1.816	7.4
Cardiff	1 GHz Intel Pentium-III	758.9	0.307	0.614	2.5
Brno	2.4 GHz Intel Xeon	152.8	1.525	12.200	49.5
Berlin	500 MHz MIPS R14000	3701.4	0.062	0.992	4.0
total				24.668	100.0

As a benchmark, we first ran the parallel version of the ray tracer with a smaller problem size (512×512 , with 24 bit color) on a single machine on all clusters. This way, we can compute the relative speeds of the different machines and JVMs. The results are presented in Table 4.4. To calculate the relative speed of each machine/JVM combination, we normalized the run times relative to the run time of the ray tracer on a node of

the DAS-2 cluster in Amsterdam. It is interesting to note that the quality of the JIT compiler can have a large impact on the performance at the application level. A node in the DAS-2 cluster and the machine in Cardiff are both 1 GHz Intel Pentium-IIIs, but there is more than a factor of three difference in application performance. This is caused by the different JIT compilers that were used. On the DAS-2, we used the more efficient IBM 1.4 JIT, while the SUN 1.4 JIT (HotSpot) was installed on the machine in Cardiff.

Furthermore, the results show that, although the clock frequency of the machine at Brno is 2.4 times as high as the frequency of a DAS-2 node, the speed improvement is only 53%. Both machines use Intel processors, but the Xeon machine in Brno is based on Pentium-4 processors, which do less work per cycle than the Pentium-III CPUs that are used by the DAS-2. We have to conclude that it is in general not possible to simply use the clock frequencies to compare processor speeds.

Finally, it is obvious that the Origin machine in Berlin is slow compared to the other machines. This is partly caused by the inefficient JIT, which is based on the SUN HotSpot JVM. Because of the combination of slow processors and the inefficient JIT, the 16 nodes of the Origin we used are about as fast as a single 1 GHz Pentium-III with the IBM JIT. The Origin thus hardly contributes anything to the computation. The table shows that, although we used 40 CPUs in total for the grid run, the relative speed of these processors together adds up to 24.668 DAS-2 nodes (1 GHz Pentium-IIIs). The percentage of the total compute power that each individual cluster delivers is shown in the rightmost column of Table 4.4.

TABLE 4.5
Performance of the ray tracer application on the GridLab testbed.

algorithm	run time (s)	communication time (s)	overhead	parallelization time (s)	overhead	efficiency
<i>nighttime</i>						
RS	877.6	198.5	36.1%	121.9	23.5%	62.6%
CRS	676.5	35.4	6.4%	83.9	16.6%	81.3%
<i>daytime</i>						
RS	2083.5	1414.5	257.3%	111.8	21.7%	26.4%
CRS	693.0	40.1	7.3%	95.7	18.8%	79.3%
<i>single cluster 25</i>						
RS	579.6	11.3	2.0%	11.0	1.9%	96.1%

We also ran the ray tracer on a single DAS-2 machine, with the large problem size that we will use for the grid runs. This took 13746 seconds (almost four hours). The sequential program without the Satin constructs takes 13564 seconds, the overhead of the parallel version thus is about 1%. With perfect speedup, the run time of the parallel program on the GridLab testbed would be 13564 divided by 24.668, which is 549.8 seconds (about nine minutes). We consider this run time the upper bound on the performance that can be achieved on the testbed, $t_{perfect}$. We can use this number to calculate the efficiency that is achieved by the real parallel runs. We call the actual run time of the application on the testbed t_{grid} . In analogy to Section 2.3, efficiency can be defined as follows:

$$efficiency = \frac{t_{perfect}}{t_{grid}} * 100\%$$

We have also measured the time that is spent in communication (t_{comm}). This includes idle time, because all idle time in the system is caused by waiting for communication to finish. We calculate the relative communication overhead with this formula:

$$communication\ overhead = \frac{t_{comm}}{t_{perfect}} * 100\%$$

Finally, the time that is lost due to parallelization overhead (t_{par}) is calculated as shown below:

$$t_{par} = t_{grid} - t_{comm} - t_{perfect}$$

$$parallelization\ overhead = \frac{t_{par}}{t_{perfect}} * 100\%$$

TABLE 4.6
Communication statistics for the ray tracer application on the GridLab testbed.

alg.	intra cluster		inter cluster	
	messages	MByte	messages	MByte
<i>nighttime</i>				
RS	3218	41.8	11473	137.3
CRS	1353295	131.7	12153	86.0
<i>daytime</i>				
RS	56686	18.9	149634	154.1
CRS	2148348	130.7	10115	82.1
<i>single cluster 25</i>				
RS	45458	155.6	n.a.	n.a.

The results of the grid runs are shown in Table 4.5. For reference, we also provide measurements on a single cluster, using 25 nodes of the DAS-2 system. The results presented here are the fastest runs out of three experiments. During daytime, the performance of the ray tracer with RS showed a large variability, some runs took longer than an hour to complete, while the fastest run took about half an hour. Therefore, in this particular case, we took the best result of six runs. This approach thus is in favor of RS. With CRS, this effect does not occur: the difference between the fastest and the slowest run during daytime was less than 20 seconds. During night, when there is little Internet traffic, the application with CRS is already more than 200 seconds faster (about 23%) than with the RS algorithm. During daytime, when the Internet links are heavily used, CRS outperforms RS by a factor of three. Regardless of the time of the day, the efficiency of a parallel run with CRS is about 80%.

The numbers in Table 4.5 show that the parallelization overhead on the testbed is significantly higher compared to a single cluster. Sources of this overhead are thread creation and switching caused by incoming steal requests, and the locking of the work queues. The overhead is higher on the testbed, because five of the six machines we use are SMPs (i.e. they have a shared memory architecture). In general, this means that the CPUs in such a system have to share resources, making memory access and especially synchronization potentially more expensive. The latter has a negative effect on the performance of the work queues. Also, multiple CPUs share a single network interface, making access to the communication device more expensive. The current implementation of Satin treats SMPs as clusters (i.e., on a N -way SMP, we start N JVMs). Therefore, Satin pays the price of the SMP overhead, but does not exploit the benefits of SMP systems, such as the available shared memory. An implementation that does utilize shared memory when available is planned for the future.

Communication statistics of the grid runs are shown in Table 4.6. The numbers in the table totals for the whole run, summed over all CPUs. Again, statistics for a single cluster run are included for reference. The numbers show that almost all of the overhead of RS is in excessive wide-area communication. During daytime, for instance, it tries to send 154 MByte over the busy Internet links. During the time-consuming wide-area transfers, the sending machine is idle, because the algorithm is synchronous. CRS sends only about 82 MBytes over the wide-area links (about half the amount of RS), but more importantly, the transfers are asynchronous. With CRS, the machine that initiates the wide-area traffic concurrently tries to steal work in the local cluster, and also concurrently executes the work that is found.

CRS effectively trades less wide-area traffic for more local communication. As shown in Table 4.6, the run during the night sends about 1.4 million local-area messages. During daytime, the CRS algorithm has to do more effort to keep the load balanced: during the wide-area steals, about 2.1 million local messages are sent while trying to find work within the local clusters. This is about 60% more than during the night. Still, only 40.1 seconds are spent communicating. With CRS, the run during daytime only takes 16.5 seconds (about 2.4%) longer than the run at night. The total communication overhead of CRS is at most 7.3%, while with RS, this can be as much as two thirds of the run time (i.e. the algorithm spends more time on communicating than on calculating useful work).

Because all idle time is caused by communication, the time that is spent on the actual computation can be calculated by subtracting the communication time from the actual run time (t_{grid}). Because we have gathered the communication statistics per machine (not shown), we can calculate the total time a whole *cluster* spends

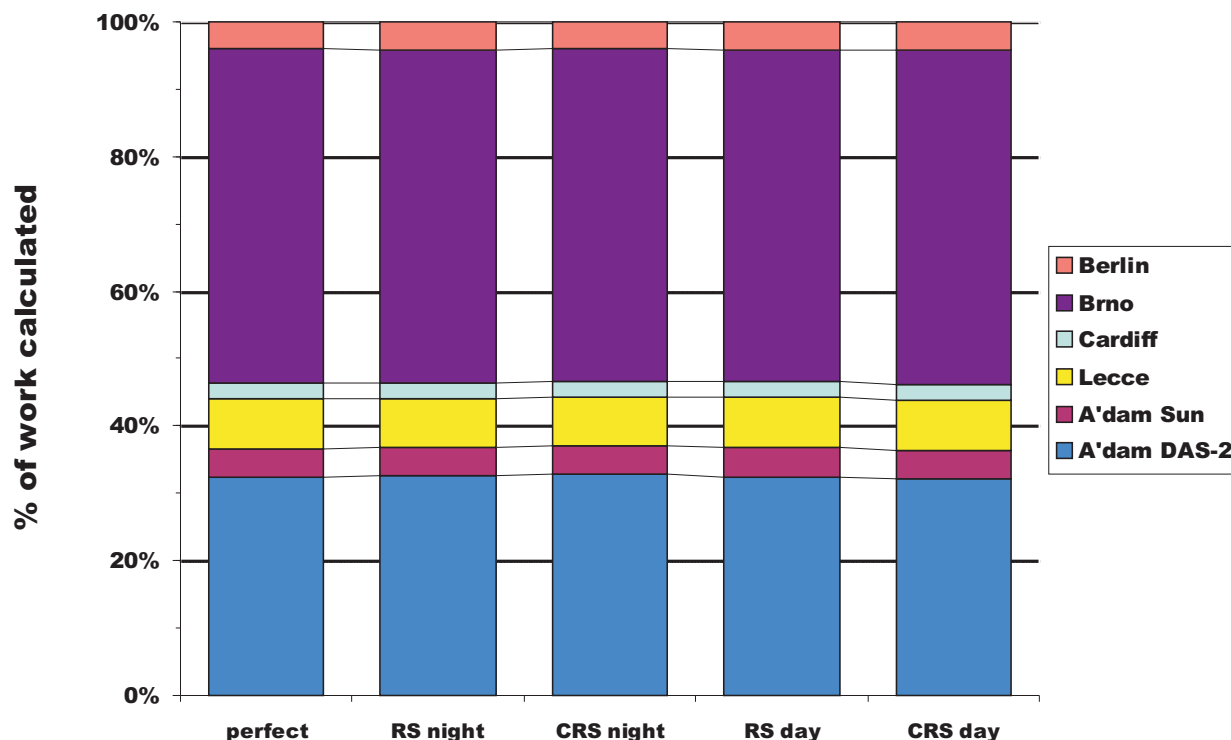


FIG. 4.2. Distribution of work over the different sites.

computing the actual problem. Given the amount of time a cluster performs useful work and the relative speed of the cluster, we can calculate what fraction of the total work is calculated by each individual cluster. We can compare this workload distribution with the ideal distribution which is represented by the rightmost column of Table 4.4. The ideal distribution and the results for the four grid runs are shown in Figure 4.2. The difference between the perfect distribution and the actual distributions of the four grid runs is hardly visible. From the figure, we can conclude that, although the workload distribution of both RS and CRS is virtually perfect, the RS algorithm itself spends a large amount of time on *achieving* this distribution. CRS does not suffer from this problem, because wide-area traffic is asynchronous and is overlapped with useful work that was found locally. Still, it achieves an almost optimal distribution.

To summarize, the experiment described in this section shows that the Java-centric approach to grid computing, and the Satin/Ibis system in particular, works extremely well in practice in a real grid environment. It took hardly any effort to run Ibis and Satin on a heterogeneous system. Furthermore, the performance results clearly show that CRS outperforms RS in a real grid environment, especially when the wide-area links are also used for other (Internet) traffic. With CRS, the system is idle (waiting for communication) during only a small fraction of the total run time. We expect even better performance when larger clusters are used, as indicated by our simulator results from Section 2.3.

5. Related work. We have discussed a Java-centric approach to writing wide-area parallel (grid computing) applications. Most other grid computing systems (e.g., Globus [10] and Legion [13]) support a variety of languages. GridLab [2] is building a toolkit of grid services that can be accessed from various programming languages. Converse [15] is a framework for multi-lingual interoperability. The SuperWeb [1], and Bayanihan [29] are examples of global computing infrastructures that support Java. A language-centric approach makes it easier to deal with heterogeneous systems, since the data types that are transferred over the networks are limited to the ones supported in the language (thus obviating the need for a separate interface definition language) [32].

The AppLeS (short for application-level scheduling) project provides a framework for adaptively scheduling

applications on the grid [5]. AppLeS focuses on selecting the best set of resources for the application out of the resource pool of the grid. Satin addresses the more low-level problem of load balancing the parallel computation itself, given some set of grid resources. AppLeS provides (amongst others) a template for master-worker applications, whereas Satin provides load balancing for the more general class of divide-and-conquer algorithms.

Many divide-and-conquer systems are based on the C language. Among them, Cilk [7] only supports shared-memory machines, CilkNOW [9] and DCPAR [12] run on local-area, distributed-memory systems. SilkRoad [27] is a version of Cilk for distributed memory systems that uses a software DSM to provide shared memory to the programmer, targeting at small-scale, local-area systems.

The Java classes presented by Lea [18] can be used to write divide-and-conquer programs for shared-memory systems. Satin is a divide-and-conquer extension of Java that was designed for wide-area systems, without shared memory. Like Satin, Javar [6] is compiler-based. With Javar, the programmer uses annotations to indicate divide-and-conquer and other forms of parallelism. The compiler then generates multithreaded Java code, that runs on any JVM. Therefore, Javar programs run only on shared-memory machines and DSM systems.

Herrmann et al. [14] describe a compiler-based approach to divide-and-conquer programming that uses skeletons. Their DHC compiler supports a purely functional subset of Haskell, and translates source programs into C and MPI. Alt et al. [3] developed a Java-based system, in which skeletons are used to express parallel programs, one of which for expressing divide-and-conquer parallelism. Although the programming system targets grid platforms, it is not clear how scalable the approach is: in [3], measurements are provided only for a local cluster of 8 machines.

Most systems described above use some form of random stealing (RS). It has been proven [8] that RS is optimal in space, time and communication, at least for relatively tightly coupled systems like SMPs and clusters that have homogeneous communication performance. In previous work [26], we have shown that this property cannot be extended to wide-area systems. We extended RS to perform asynchronous wide-area communication interleaved with synchronous local communication. The resulting randomized algorithm, called CRS, does perform well in loosely-coupled systems.

Another Java-based divide-and-conquer system is Atlas [4]. Atlas is a set of Java classes that can be used to write divide-and-conquer programs. Javelin 3 [24] provides a set of Java classes that allow programmers to express branch-and-bound computations, such as the traveling salesperson problem. Like Satin, Atlas and Javelin 3 are designed for wide-area systems. Both Atlas and Javelin 3 use tree-based hierarchical scheduling algorithms. We found that such algorithms are inefficient for fine-grained applications and that CRS performs better [26].

6. Conclusions. Grid programming environments need to be both *portable* and *efficient* to exploit the computational power of dynamically available resources. Satin makes it possible to write divide-and-conquer applications in Java, and is targeted at clustered wide-area systems. The Satin implementation on top of our new Ibis platform combines Java's *run everywhere* with efficient communication between JVMs. The resulting system is easy to use in a grid environment. To achieve high performance, Satin uses a special grid-aware load-balancing algorithm. Previous simulation results suggested that this algorithm is more efficient than traditional algorithms that are used on tightly-coupled systems. In this paper, we verified these simulation results in a real grid environment.

We evaluated Satin/Ibis on the highly heterogeneous testbed of the EU-funded GridLab project, showing that Satin's load-balancing algorithm automatically adapts both to heterogeneous processor speeds and varying network performance, resulting in efficient utilization of the computing resources. Measurements show that Satin's CRS algorithm indeed outperforms the widely used RS algorithm by a wide margin. With CRS, Satin achieves around 80% efficiency, even during daytime when the links between the sites are heavily loaded. In contrast, with the traditional RS algorithm, the efficiency drops to about 26% when the wide-area links are congested.

Acknowledgments. Part of this work has been supported by the European Commission, grant IST-2001-32133 (GridLab). We would also like to thank Olivier Aumage, Rutger Hofman, Cerial Jacobs, Maik Nijhuis and Gosia Wrzesińska for their contributions to the Ibis code. Kees Verstoep is doing a marvelous job maintaining the DAS clusters. Aske Plaat suggested performing an evaluation of Satin on a real grid testbed. John Romein, Matthew Shields and Massimo Cafaro gave valuable feedback on this manuscript.

REFERENCES

- [1] A. D. ALEXANDROV, M. IBEL, K. E. SCHAUSER, AND C. J. SCHEIMAN, *SuperWeb: Research Issues in Java-Based Global Computing*, Concurrency: Practice and Experience, 9 (1997), pp. 535–553.
- [2] G. ALLEN, K. DAVIS, K. N. DOLKAS, N. D. DOULAMIS, T. GOODALE, T. KIELMANN, A. MERZKY, J. NABRZYSKI, J. PUKACKI, T. RADKE, M. RUSSELL, E. SEIDEL, J. SHALF, AND I. TAYLOR, *Enabling Applications on the Grid - A GridLab Overview*, International Journal of High Performance Computing Applications, (2003). accepted for publication.
- [3] M. ALT, H. BISCHOF, AND S. GORLATCH, *Program Development for Computational Grids using Skeletons and Performance Prediction*, Parallel Processing Letters, 12 (2002), pp. 157–174. World Scientific Publishing Company.
- [4] E. J. BALDESCHWIELER, R. BLUMOFÉ, AND E. BREWER, *ATLAS: An Infrastructure for Global Computing*, in Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications, Connemara, Ireland, September 1996, pp. 165–172.
- [5] F. BERMAN, R. WOLSKI, S. FIGUEIRA, J. SCHOPF, AND G. SHAO, *Application-level Scheduling on Distributed Heterogeneous Networks*, in Proceedings of the ACM/IEEE Conference on Supercomputing (SC'96), Pittsburgh, PA, November 1996. Online at <http://www.supercomp.org>.
- [6] A. BIK, J. VILLACIS, AND D. GANNON, *Javar: A Prototype Java Restructuring Compiler*, Concurrency: Practice and Experience, 9 (1997), pp. 1181–1191.
- [7] R. D. BLUMOFÉ, C. F. JOERG, B. C. KUSZMAUL, C. E. LEISERSON, K. H. RANDALL, AND Y. ZHOU., *Cilk: An Efficient Multithreaded Runtime System*, in 5th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'95), Santa Barbara, CA, July 1995, pp. 207–216.
- [8] R. D. BLUMOFÉ AND C. E. LEISERSON, *Scheduling Multithreaded Computations by Work Stealing*, in 35th Annual Symposium on Foundations of Computer Science (FOCS '94), Santa Fe, New Mexico, November 1994, pp. 356–368.
- [9] R. D. BLUMOFÉ AND P. LISIECKI, *Adaptive and Reliable Parallel Computing on Networks of Workstations*, in USENIX 1997 Annual Technical Conference on UNIX and Advanced Computing Systems, Anaheim, CA, 1997, pp. 133–147.
- [10] I. FOSTER AND C. KESSELMAN, *Globus: A Metacomputing Infrastructure Toolkit*, International Journal of Supercomputer Applications, 11 (1997), pp. 115–128.
- [11] I. FOSTER, C. KESSELMAN, G. TSUDIK, AND S. TUECKE, *A security architecture for computational grids*, in 5th ACM Conference on Computer and Communication Security, San Francisco, CA, November 1998, pp. 83–92.
- [12] B. FREISLEBEN AND T. KIELMANN, *Automated Transformation of Sequential Divide-and-Conquer Algorithms into Parallel Programs*, Computers and Artificial Intelligence, 14 (1995), pp. 579–596.
- [13] A. GRIMSHAW AND W. A. WULF, *The Legion Vision of a Worldwide Virtual Computer*, Comm. ACM, 40 (1997), pp. 39–45.
- [14] C. A. HERRMANN AND C. LENGAUER, *HDC: A Higher-Order Language for Divide-and-Conquer*, Parallel Processing Letters, 10 (2000), pp. 239–250.
- [15] L. V. KALÉ, M. BHANDARKAR, N. JAGATHESAN, S. KRISHNAN, AND J. YELON, *Converse: An interoperable framework for parallel programming*, in Intl. Parallel Processing Symposium, 1996.
- [16] K. KANEDA, K. TAURA, AND A. YONEZAWA, *Virtual private grid: A command shell for utilizing hundreds of machines efficiently*, in 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2002), Berlin, Germany, May 2002, pp. 212–219.
- [17] T. KIELMANN, H. E. BAL, J. MAASSEN, R. VAN NIEUWPOORT, L. EYRAUD, R. HOFMAN, AND K. VERSTOEP, *Programming Environments for High-Performance Grid Computing: the Albatross Project*, Future Generation Computer Systems, 18 (2002), pp. 1113–1125.
- [18] D. LEA, *A Java Fork/Join Framework*, in Proceedings of the ACM 2000 Java Grande Conference, San Francisco, CA, June 2000, pp. 36–43.
- [19] C. LEE, S. MATSUOKA, D. TALIA, A. SUSSMANN, M. MÜLLER, G. ALLEN, AND J. SALTZ, *A Grid programming primer*. Global Grid Forum, August 2001.
- [20] M. LEECH, M. GANIS, Y. LEE, R. KURIS, D. KOBLAS, AND L. JONES, *RFC 1928: SOCKS protocol version 5*, April 1996.
- [21] J. MAASSEN, T. KIELMANN, AND H. BAL, *GMI: Flexible and Efficient Group Method Invocation for Parallel Programming*, in In proceedings of LCR-02: Sixth Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers, Washington DC, March 2002, pp. 1–6.
- [22] J. MAASSEN, T. KIELMANN, AND H. E. BAL, *Parallel Application Experience with Replicated Method Invocation*, Concurrency and Computation: Practice and Experience, 13 (2001), pp. 681–712.
- [23] J. MAASSEN, R. VAN NIEUWPOORT, R. VELDEMA, H. BAL, T. KIELMANN, C. JACOBS, AND R. HOFMAN, *Efficient Java RMI for Parallel Programming*, ACM Transactions on Programming Languages and Systems, 23 (2001), pp. 747–775.
- [24] M. O. NEARY AND P. CAPPELLO, *Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing*, in Proceedings of the Joint ACM 2002 Java Grande - ISCOPE (International Symposium on Computing in Object-Oriented Parallel Environments) Conference, Seattle, November 2002, pp. 56–65.
- [25] *Public netperf homepage*. www.netperf.org.
- [26] R. V. VAN NIEUWPOORT, T. KIELMANN, AND H. E. BAL, *Efficient Load Balancing for Wide-area Divide-and-Conquer Applications*, in Proceedings Eighth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'01), Snowbird, UT, June 2001, pp. 34–43.
- [27] L. PENG, W. WONG, M. FENG, AND C. YUEN, *SilkRoad: A Multithreaded Runtime System with Software Distributed Shared Memory for SMP Clusters*, in IEEE International Conference on Cluster Computing (Cluster2000), Chemnitz, Saxony, Germany, November 2000, pp. 243–249.
- [28] M. PHILIPPSEN, B. HAUMACHER, AND C. NESTER, *More efficient serialization and RMI for Java*, Concurrency: Practice and Experience, 12 (2000), pp. 495–518.
- [29] L. F. G. SARMENTA, *Volunteer Computing*, PhD thesis, Dept. of Electrical Engineering and Computer Science, MIT, 2001.
- [30] Y. TANAKA, H. NAKADA, S. SEKIGUCHI, T. SUZUMURA, AND S. MATSUOKA, *Ninf-G: A Reference Implementation of RPC-based Programming Middleware for Grid Computing*, Journal of Grid Computing, 1 (2003), pp. 41–51.

- [31] R. V. VAN NIEUWPOORT, J. MAASSEN, R. HOFMAN, T. KIELMANN, AND H. E. BAL, *Ibis: an Efficient Java-based Grid Programming Environment*, in Joint ACM Java Grande - ISCOPE 2002 Conference, Seattle, Washington, USA, November 2002, pp. 18–27.
- [32] A. WOLLRATH, J. WALDO, AND R. RIGGS, *Java-Centric Distributed Computing*, IEEE Micro, 17 (1997), pp. 44–53.
- [33] I.-C. WU AND H. KUNG, *Communication Complexity for Parallel Divide-and-Conquer*, in 32nd Annual Symposium on Foundations of Computer Science (FOCS '91), San Juan, Puerto Rico, Oct. 1991, pp. 151–162.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 15, 2003.

Accepted: September 1, 2003.



RUN-TIME ADAPTATION OF GRID DATA PLACEMENT JOBS

G. KOLA*, T. KOSAR* & M. LIVNY*

Abstract. Grid presents a continuously changing environment. It also introduces a new set of failures. The data grid initiative has made it possible to run data-intensive applications on the grid. Data-intensive grid applications consist of two parts: a data placement part and a computation part. The data placement part is responsible for transferring the input data to the compute node and the result of the computation to the appropriate storage system. While work has been done on making computation adapt to changing conditions, little work has been done on making the data placement adapt to changing conditions. In this work, we have developed an infrastructure which observes the environment and enables run-time adaptation of data placement jobs. We have enabled Stork, a scheduler for data placement jobs in heterogeneous environments like the grid, to use this infrastructure and adapt the data placement job to the environment just before execution. We have also added dynamic protocol selection and alternate protocol fall-back capability to Stork to provide superior performance and fault tolerance.

Key words. Grid, data placement, run-time adaptation, scheduling, data intensive applications, dynamic protocol selection, stork, condor.

1. Introduction. The grid [10] [11] [19] presents a continuously changing environment. The data grid initiative has increased the underlying network capacity and enabled running of data-intensive applications on the grid. Data-intensive applications consist of two parts: a data placement part and a computation part. The data placement part is responsible for transferring the input data to the compute node and the result of the computation to the appropriate storage system. Data placement encompasses all data movement related activities such as transfer, staging, replication, data positioning, space allocation and deallocation. While work has been done on making computation adapt to changing conditions, little work has been done on making the data placement adapt to changing conditions.

Sophisticated protocols developed for grid data transfers like GridFTP [1] allow tuning depending on the environment to achieve the best performance. While tuning by itself is difficult, it is further complicated by the changing environment. The parameters which are optimal at the time of job submission, may no longer be optimal at the time of execution. The best time to tune the parameters is just before execution of the data placement job. Determining the environment characteristics and performing tuning for each job may impose a significant overhead. Ideally, we need an infrastructure that detects environmental changes and performs appropriate tuning and uses the tuned parameters for subsequent data placement jobs.

Many times, we have the ability to use different protocols for data transfers, with each having different network, CPU and disk characteristics. The new fast protocols do not work all the time. The main reason is the presence of bugs in the implementation of the new protocols. The more robust protocols work for most of the time but do not perform as well. This presents a dilemma to the users who submit data placement jobs to data placement schedulers. If they choose the fast protocol, some of their transfers may never complete and if they choose the slower protocol, their transfer would take a very long time. Ideally users would want to use the faster protocol when it works and switch to the slower more reliable protocol when the fast one fails. Unfortunately, when the fast protocol would fail is not known apriori. The decision on which protocol to use is best done just before starting the transfer.

Some users simply want data transferred and do not care about the protocol being used. Others have some preference such as: as fast as possible, as low a CPU load as possible, as minimal memory usage as possible. The machines where the jobs are being executed may have some characteristics which might favor some protocol. Further the machine characteristics may change over time due to hardware and software upgrades. Most users do not understand the performance characteristics of the different protocols and inevitably end up using a protocol that is known to work. In case of failures, they just wait for the failure to be fixed, even though other protocols may be working.

An ideal system is one that allows normal users to specify their preference and chooses the appropriate protocol based on their preference and machine characteristics. It should also switch to the next most appropriate protocol in case the current one stops working. It should also allow sophisticated users to specify the protocol to use and the alternate protocols in case of failure. Such a system would not only reduce the complexity of

*Department of Computer Sciences, University of Wisconsin-Madison, 1210 W. Dayton St. Madison, WI 53706, USA. ({kola, kosart, miron}@cs.wisc.edu).

programming the data transfer but also provide superior failure recovery strategy. The system may also be able to improve performance because it can perform on-the-fly optimization.

In this work, we have developed a monitoring infrastructure which determines the environment characteristics and detects any subsequent change. The environment characteristics are used by the tuning infrastructure to generate tuned parameters for the various protocols. These tuned parameters are fed to a data placement scheduler. The data placement scheduler uses the tuned parameters while executing the data placement jobs submitted to it, essentially performing run-time adaptation of data placement jobs. We have also added dynamic protocol selection and alternate protocol fall-back capability to our prototype data placement scheduler. Dynamic protocol selection determines the protocols that are available on a particular host and uses an appropriate protocol for data transfer between any two hosts. Alternate protocol fall-back allows the data placement scheduler to switch to a different protocol if the protocol being used for a transfer stops working.

2. Related Work. Network Weather Service (NWS) [25] is a distributed system which periodically gathers readings from network and CPU resources, and uses numerical models to generate forecasts for a given time frame. Vazhkudai [24] found that the network throughput predicted by NWS was much less than the actual throughput achieved by GridFTP. He attributed the reason for it being that NWS by default was using 64KB data transfer probes with normal TCP window size to measure throughput. We wanted our network monitoring infrastructure to be as accurate as possible and wanted to use it to tune protocols like GridFTP.

Semke [20] introduces automatic TCP buffer tuning. Here the receiver is expected to advertise large enough windows. Fisk [9] points out the problems associated with [20] and introduces dynamic right sizing which changes the receiver window advertisement according to estimated sender congestion window. 16-bit TCP window size field and 14-bit window scale option which needs to be specified during connection setup, introduce more complications. While a higher value of the window-scale option allows a larger window, it increases the granularity of window increments and decrements. While large data transfers benefit from large window size, web and other traffic are adversely affected by the larger granularity of window-size changes.

Linux 2.4 kernel used in our machines implements dynamic right-sizing, but the receiver window size needs to be set explicitly if a window size large than 64 KB is to be used. Autobuf [15] attempts to tune TCP window size automatically by performing bandwidth estimation before the transfer. Unfortunately there is no negotiation of TCP window size between server and client which is needed for optimal performance. Also performing a bandwidth estimation before every transfer introduces too much of an overhead.

Fearman et. al [8] introduce the Adaptive Regression Modeling (ARM) technique to forecast data transfer times for network-bound distributed data-intensive applications. Ogura et. al [17] try to achieve optimal bandwidth even when the network is under heavy contention, by dynamically adjusting transfer parameters between two clusters, such as the number of socket stripes and the number of network nodes involved in transfer.

In [5], Carter et. al. introduce tools to estimate the maximum possible bandwidth along a given path, and to calculate the current congestion along a path. Using these tools, they demonstrate how dynamic server selection can be performed to achieve application-level congestion avoidance.

Thain et. al. propose the Ethernet approach [21] to Grid Computing, in which they introduce a simple scripting language which can handle failures in a manner similar to exceptions in some languages. The Ethernet approach is not aware of the semantics of the jobs it is running, its duty is retrying any given job for a number of times in a fault tolerant manner. Kangaroo [22] tries to achieve high throughput by making opportunistic use of disk and network resources.

Application Level Schedulers (AppLeS) [4] have been developed to achieve efficient scheduling by taking into account both application-specific and dynamic system information. AppLeS agents use dynamic system information provided by the NWS.

Beck et. al. introduce Logistical Networking [2] which performs global scheduling and optimization of data movement, storage and computation based on a model that takes into account all the network's underlying physical resources.

3. Methodology. The environment in which data placement jobs execute keeps changing all the time. The network bandwidth keeps fluctuating. The network route changes once in a while. The optic fiber may get upgraded increasing the bandwidth. New disks and raid-arrays may be added to the system. The monitoring and tuning infrastructure monitors the environment and tunes the different parameters accordingly. The data placement scheduler then uses these tuned parameters to intelligently schedule and execute the transfers.

Figure 3.1 shows the components of the monitoring and tuning infrastructure and the interaction with the data placement scheduler.

3.1. Monitoring Infrastructure. The monitoring infrastructure monitors the disk, memory and network characteristics. The infrastructure takes into account that the disk and memory characteristics change less frequently and the network characteristics change more frequently. The disk and memory characteristics are measured once after the machine is started. If a new disk is added on the fly (hot-plugin), there is an option to inform the infrastructure to determine the characteristics of that disk. The network characteristics are measured periodically. The period is tunable. If the infrastructure finds that the network characteristics are constant for a certain number of measurements, it reduces the frequency of measurement till a specified minimum is reached. The objective of this is to keep the overhead of measurement as low as possible.

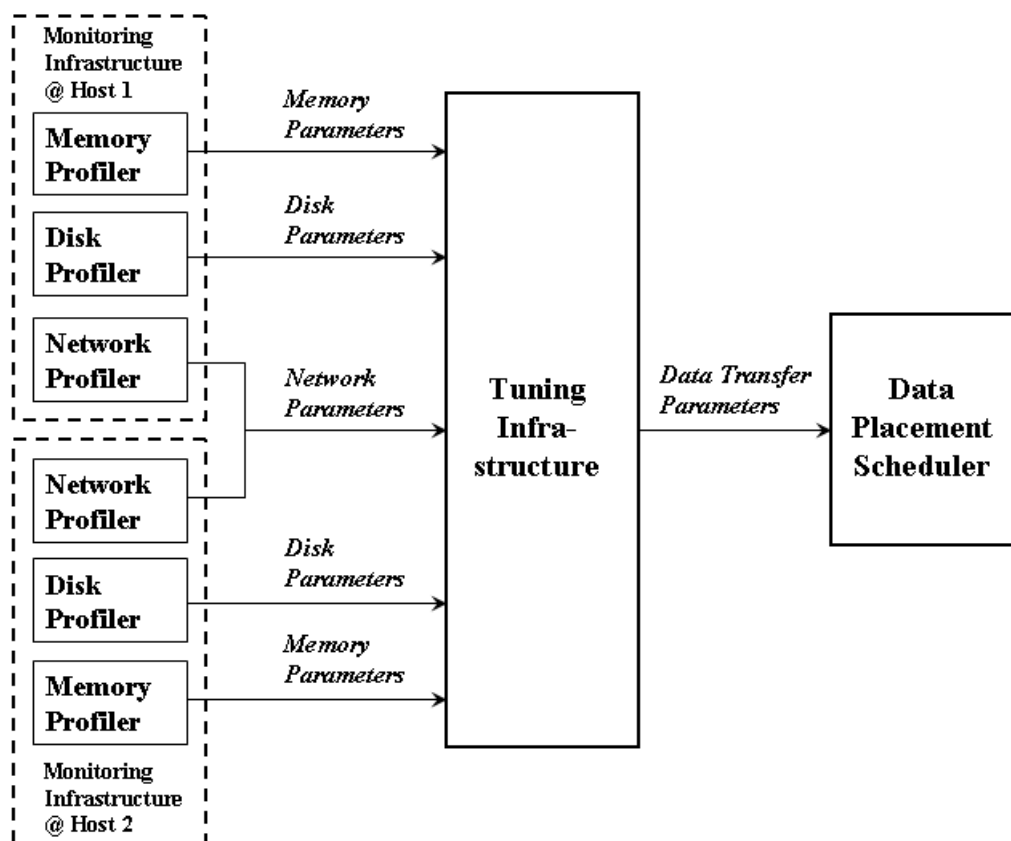


FIG. 3.1. *Monitoring and Tuning Infrastructure.* This figure shows an overview of the monitoring and tuning infrastructure. The different profilers determine the various environment conditions and the tuning infrastructure uses that information to generate optimal parameter values.

The disk and memory characteristics are determined by intrusive techniques, and the network characteristics are determined by a combination of intrusive and non-intrusive techniques. The memory characteristic of interest to us is the optimal memory block size to be used for memory-to-memory copy. The disk characteristics measured include the optimal read and write block sizes and the incremental block size that can be added to the optimal value to get the same performance.

The network characteristics measured are the following: end-to-end bandwidth, end-to-end latency, number of hops, the latency of each hop and kernel TCP parameters. Since end-to-end measurement requires two hosts, this measurement is done between every pair of hosts that may transfer data between each other. The end-to-end bandwidth measurement uses both intrusive and non-intrusive techniques. The non-intrusive technique uses packet dispersion technique to measure the bandwidth. The intrusive technique performs actual transfers. First, the non-intrusive technique is used and the bandwidth is determined. Then actual transfer is performed to measure the end-to-end bandwidth. If the numbers widely differ, the infrastructure performs a certain number

of both of the network measurements and finds the correlation between the two. After this initial setup, a light-weight network profiler is run which uses only non-intrusive measuring technique. While we perform a longer initial measurement for higher accuracy, the subsequent periodic measurements are very light-weight and do not perturb the system.

3.2. Tuning Infrastructure. The tuning infrastructure uses the information collected by monitoring infrastructure and tries to determine the optimal I/O block size, TCP buffer size and the number of TCP streams for the data transfer from a given node X to a given node Y. The tuning infrastructure has the knowledge to perform protocol-specific tuning. For instance, GridFTP takes as input only a single I/O block size, but the source and destination machines may have different optimal I/O block sizes. For such cases, the tuning finds the I/O block size which is optimal for both of them. The incremental block size measured by the disk profiler is used for this. The tuning infrastructure feeds the data transfer parameters to the data placement scheduler.

3.3. Scheduling Data Transfers. The data placement scheduler uses the information provided by the tuning infrastructure to make intelligent decisions for scheduling and executing the data placement jobs.

In our study, we used the Stork [13] data placement scheduler to monitor, manage, and schedule the data transfers over the wide area network. Stork is a specialized scheduler for data placement activities in heterogeneous environments. Stork can queue, schedule, monitor and manage data placement jobs, and it ensures that the jobs complete.

Stork is aware of the semantics of the data placement requests submitted to it, so it can make intelligent scheduling decisions with regard to each individual request. For example, if a transfer of a large file fails, Stork can transfer only parts of the file not already transferred. We have made some enhancements to Stork that enable it to adaptively schedule data transfers at run-time using the information provided by monitoring and tuning infrastructure. These enhancements include dynamic protocol selection and run-time protocol auto-tuning. The details of these enhancements are discussed in section 5.

4. Implementation. We have developed a set of tools to determine disk, memory and network characteristics and using those values determine the optimal parameter values to be used for data transfers. We executed these tools in a certain order and fed the results to Stork data placement scheduler which then performed run-time adaptation of the wide-area data placement jobs submitted to it.

4.1. Disk and Memory Profilers. The disk profiler determines the optimal read and write block sizes and the increment that can be added to the optimal block size to get the same performance. A list of pathnames and the average file size is fed to the disk profiler. So, in a multi-disk system, the mount point of the different disks are passed to the disk profiler. In the case of a raid-array, the mount point of the raid array is specified. For each of the specified paths, the disk profiler finds the optimal read and write block size and the optimal increment that can be applied to these block sizes to get the same performance. It also lists the read and write disk bandwidths achieved by the optimal block sizes.

For determining the optimal write block size, the profiler creates a file in the specified path and writes the average file size of data in block-size chunks and flushes the data to disk at the end. It repeats the experiment for different block sizes and finds the optimal. For determining the read block size, it uses the same technique except that it flushes the kernel buffer cache to prevent cache effects before repeating the measurement for a different block size. Since normal kernels do not allow easy flushing of the kernel buffer cache, the micro-benchmark reads in a large dummy file of size greater than the buffer cache size essentially flushing it. The memory profiler finds the maximum memory-to-memory copy bandwidth and the block size to be used to achieve it.

4.2. Network Profiler. The network profiler gets the kernel TCP parameters from `/proc`. It runs Pathrate [7] between given pair of nodes and gets the estimated bottleneck bandwidth and the average round-trip time. It then runs traceroute between the nodes to determine the number of hops between the nodes and the hop-to-hop latency. The bandwidth estimated by Pathrate is verified by performing actual transfers by a data transfer tool developed as part of the DiskRouter project [12]. If the two numbers differ widely, then a specified number of actual transfers and Pathrate bandwidth estimations are done to find the correlation between the two. Tools like Iperf [16] can also be used instead of the DiskRouter data transfer tool to perform the actual transfer. From experience, we found Pathrate to be the most reliable of all the network bandwidth estimation tools that use packet dispersion technique and we always found a correlation between the value returned by Pathrate

and that observed by performing actual transfer. After the initial network profiling, we run a light-weight network profiler periodically. The light-weight profiler runs only Pathrate and traceroute.

4.3. Parameter Tuner. The parameter tuner gets the information generated by the different tools and finds the optimal value of the parameters to be used for data transfer from a node X to a node Y.

To determine the optimal number of streams to use, the parameter tuner uses a simple heuristic. It finds the number of hops between the two nodes that have a latency greater than 10 ms. For each such hop, it adds an extra stream. Finally, if there are multiple streams and the number of streams is odd, the parameter tuner rounds it to an even number by adding one. The reason for doing this is that some protocols do not work well with odd number of streams. The parameter tuner calculates the bandwidth-delay product and uses that as the TCP buffer size. If it finds that it has to use more than one stream, it divides the TCP buffer size by the number of streams. The reason for adding a stream for every 10 ms hop is as follows: In a high-latency multi-hop network path, each of the hops may experience congestion independently. If a bulk data transfer using a single TCP stream occurs over such a high-latency multi-hop path, each congestion event would shrink the TCP window size by half. Since this is a high-latency path, it would take a long time for the window to grow, with the net result being that a single TCP stream would be unable to utilize the full available bandwidth. Having multiple streams reduces the bandwidth reduction of a single congestion event. Most probably only a single stream would be affected by the congestion event and halving the window size of that stream alone would be sufficient to eliminate congestion. The probability of independent congestion events occurring increases with the number of hops. Since only the high-latency hops have a significant impact because of the time taken to increase the window size, we added a stream for all high-latency hops and empirically found that hops with latency greater than 10 ms fell into the high-latency category. Note that we set the total TCP buffer size to be equal to the bandwidth delay product, so in steady state case with multiple streams, we would not be causing congestion.

The Parameter Tuner understands kernel TCP limitations. Some machines may have a maximum TCP buffer size limit less than the optimal needed for the transfer. In such a case, the parameter tuner uses more streams so that their aggregate buffer size is equal to that of the optimal TCP buffer size.

The Parameter Tuner gets the different optimal values and generates overall optimal values. It makes sure that the disk I/O block size is at least equal to the TCP buffer size. For instance, the optimal disk block size may be 1024 KB and the increment value may be 512 KB (performance of optimal + increment is same as optimal) and the optimal TCP buffer size may be 1536KB. In this case, the parameter tuner will make the protocol use a disk block size of 1536 KB and a TCP buffer size of 1536 KB. This is a place where the increment value generated by the disk profiler is useful.

The Parameter Tuner understands different protocols and performs protocol specific tuning. For example, globus-url-copy, a tool used to move data between GridFTP servers, allows users to specify only a single disk block size. The read disk block size of the source machine may be different from the write disk block size of the destination machine. In this case, the parameter tuner understands this and chooses an optimal value that is optimal for both the machines.

4.4. Coordinating the Monitoring and Tuning Infrastructure. The disk, memory and network profilers need to be run once at startup and the light-weight network profiler needs to be run periodically. We may also want to re-run the other profilers in case a new disk is added or any other hardware or operating system kernel upgrade. We have used the Directed Acyclic Graph Manager (DAGMan) [6] [23] to coordinate the monitoring and tuning process. DAGMan is service for executing multiple jobs with dependencies between them. The monitoring tools are run as Condor [14] jobs on respective machines. Condor provides a job queuing mechanism and resource monitoring capabilities for computational jobs. It also allows the users to specify scheduling policies and enforce priorities.

We executed the Parameter Tuner on the management site. Since the Parameter Tuner is a Condor job, we can execute it anywhere we have a computation resource. It picks up the information generated by the monitoring tools using Condor and produces the different tuned parameter values for data transfer between each pair of nodes. For example, if there are two nodes X and Y, then the parameter tuner generates two sets of parameters - one for transfer from node X to node Y and another for data transfer from node Y to node X. This information is fed to Stork which uses it to tune the parameters of data placement jobs submitted to it. The DAG coordinating the monitoring and tuning infrastructure is shown in Figure 4.1.

We can run an instance of parameter tuner for every pair of nodes or a certain number of pairs of nodes.

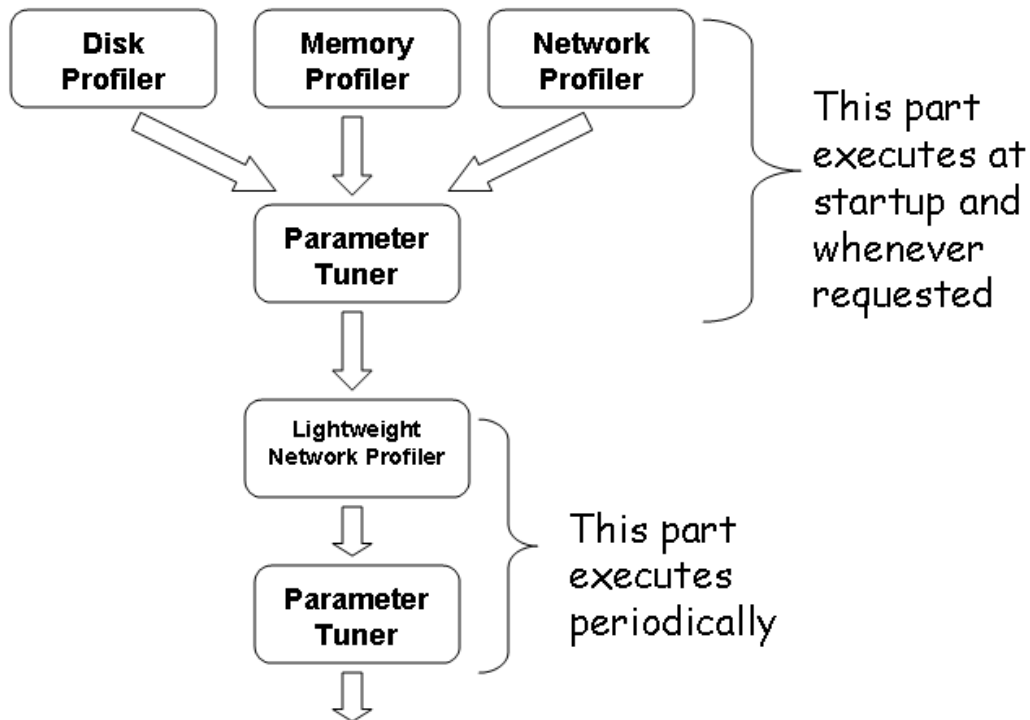


FIG. 4.1. *The DAG Coordinating the Monitoring and Tuning infrastructure. This DAG shows the order in which the monitors(profilers) and tuner are run. Initially all the profilers are run and the information is logged to persistent storage and also passed to the parameter tuner which generates the optimal parameter values. After that, the light-weight network profiler and parameter tuner are run periodically. The parameter tuner uses the values of the earlier profiler runs and the current light-weight network profiler run to generate the optimal parameter values.*

For every pair of nodes, the data fed to the parameter tuner is in the order of hundreds of bytes. Since all tools are run as Condor jobs, depending on the number of nodes involved in the transfers, we can have a certain number of parameter tuners, and they can be executed wherever there is available cycles and this architecture is not centralized with respect to the parameter tuner. In our infrastructure, we can also have multiple data placement schedulers and have the parameters for data transfers handled by a particular scheduler fed to it. In a very large system, we would have multiple data placement schedulers with each handling data movement between a certain subset of nodes.

4.5. Dynamic Protocol Selection. We have enhanced the Stork scheduler so that it can decide which data transfer protocol to use for each corresponding transfer dynamically and automatically at the run-time. Before performing each transfer, Stork makes a quick check to identify which protocols are available for both the source and destination hosts involved in the transfer. Stork first checks its own host-protocol library to see whether all of the hosts involved the transfer are already in the library or not. If not, Stork tries to connect to those particular hosts using different data transfer protocols, to determine the availability of each specific protocol on that particular host. Then Stork creates the list of protocols available on each host, and stores these lists as a library in ClassAd [18] format which is a very flexible and extensible data model that can be used to represent arbitrary services and constraints.

```

[
  host_name = "quest2.ncsa.uiuc.edu";
  supported_protocols = "diskrouter, gridftp, ftp";
]
[
  host_name = "nostos.cs.wisc.edu";
  supported_protocols = "gridftp, ftp, http";
]

```

If the protocols specified in the source and destination URLs of the request fail to perform the transfer, Stork will start trying the protocols in its host-protocol library to carry out the transfer. Stork detects a variety of protocol failures. In the simple case, connection establishment would fail and the tool would report an appropriate error code and Stork uses the error code to detect failure. In other case where there is a bug in protocol implementation, the tool may report success of a transfer, but stork would find that source and destination files have different sizes. If the same problem repeats, Stork switches to another protocol. The users also have the option to not specify any particular protocol in the request, letting Stork to decide which protocol to use at run-time.

```
[
  dap_type = "transfer";
  src_url  = "any://slic04.sdsc.edu/tmp/foo.dat";
  dest_url = "any://quest2.ncsa.uiuc.edu/tmp/foo.dat";
]
```

In the above example, Stork will select any of the available protocols on both source and destination hosts to perform the transfer. So, the users do not need to care about which hosts support which protocols. They just send a request to Stork to transfer a file from one host to another, and Stork will take care of deciding which protocol to use.

The users can also provide their preferred list of alternative protocols for any transfer. In this case, the protocols in this list will be used instead of the protocols in the host-protocol library of Stork.

```
[
  dap_type = "transfer";
  src_url  = "drouter://slic04.sdsc.edu/tmp/foo.dat";
  dest_url = "drouter://quest2.ncsa.uiuc.edu/tmp/foo.dat";
  alt_protocols = "nest-nest, gsiftp-gsiftp";
]
```

In this example, the user asks Stork to perform a transfer from slic04.sdsc.edu to quest2.ncsa.uiuc.edu using the DiskRouter protocol primarily. The user also instructs Stork to use any of the NeST [3] or GridFTP protocols in case the DiskRouter protocol does not work. Stork will try to perform the transfer using the DiskRouter protocol first. In case of a failure, it will drop to the alternative protocols and will try to complete the transfer successfully. If the primary protocol becomes available again, Stork will switch to it again. So, whichever protocol available will be used to successfully complete the user's request. In case all the protocols fail, Stork will keep trying till one of them becomes available.

4.6. Run-time Protocol Auto-tuning. Statistics for each link involved in the transfers are collected regularly and written into a file, creating a library of network links, protocols and auto-tuning parameters.

```
[
  link = "slic04.sdsc.edu - quest2.ncsa.uiuc.edu";
  protocol = "gsiftp";

  bs      = 1024KB;    //block size
  tcp_bs  = 1024KB;    //TCP buffer size
  p       = 4;        //parallelism
]
```

Before performing every transfer, Stork checks its auto-tuning library to see if there are any entries for the particular hosts involved in this transfer. If there is an entry for the link to be used in this transfer, Stork uses these optimized parameters for the transfer. Stork can also be configured to collect performance data before every transfer, but this is not recommended due to the overhead it will bring to the system.

5. Experiments and Results. We have performed two different experiments to evaluate the effectiveness of our dynamic protocol selection and run-time protocol tuning mechanisms. We also collected performance data to show the contribution of these mechanisms to wide area data transfers.

5.1. Experiment 1: Testing the Dynamic Protocol Selection. We submitted 500 data transfer requests to the Stork server running at University of Wisconsin (skywalker.cs.wisc.edu). Each request consisted of transfer of a 1.1GB image file (total 550GB) from SDSC (slic04.sdsc.edu) to NCSA (quest2.ncsa.uiuc.edu) using the DiskRouter protocol. There was a DiskRouter server installed at Starlight

(ncdm13.sl.startap.net) which was responsible for routing DiskRouter transfers. There were also GridFTP servers running on both SDSC and NCSA sites, which enabled us to use third-party GridFTP transfers whenever necessary. The experiment setup is shown in Figure 5.1.

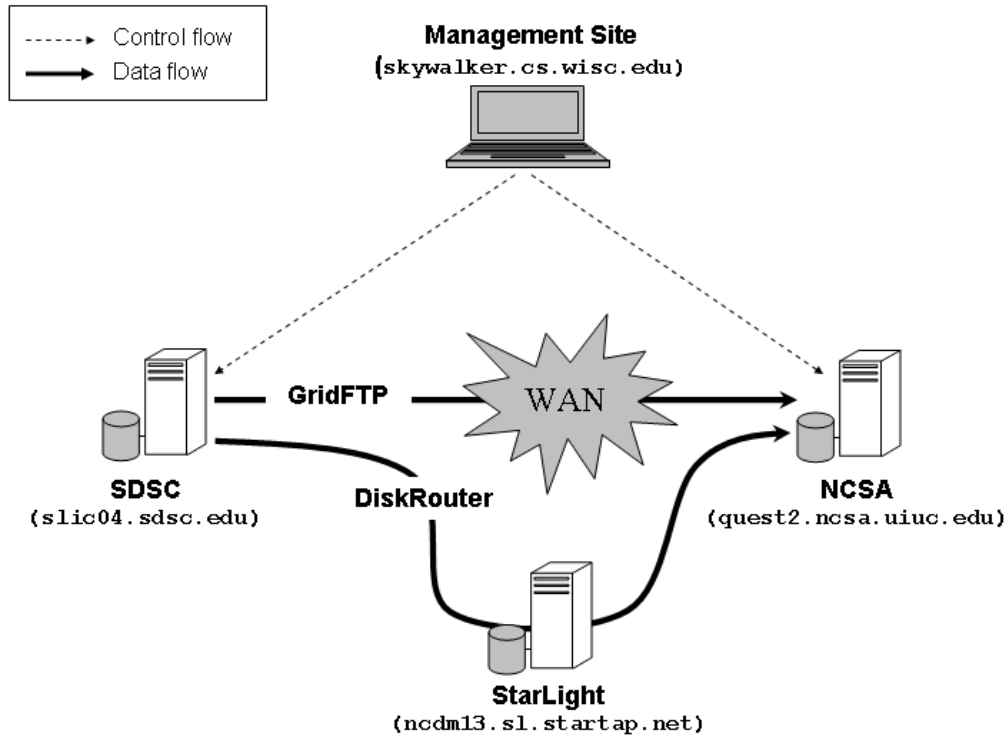


FIG. 5.1. *Experiment Setup.* DiskRouter and GridFTP protocols are used to transfer data from SDSC to NCSA. Stork was running at the Management site, and making scheduling decisions for the transfers.

At the beginning of the experiment, both DiskRouter and GridFTP services were available. Stork started transferring files from SDSC to NCSA using the DiskRouter protocol as directed by the user. After a while, we killed the DiskRouter server running at Starlight intentionally. This was done to simulate a DiskRouter server crash. Stork immediately switched the protocols and continued the transfers using GridFTP without any interruption. Switching to GridFTP caused a decrease in the performance of the transfers, as shown in Figure 5.2. The reasons of this decrease in performance is because of the fact that GridFTP does not perform auto-tuning whereas DiskRouter does. In this experiment, we set the number of parallel streams for GridFTP transfers to 10, but we did not perform any tuning of disk I/O block size or TCP buffer size. DiskRouter performs auto-tuning for the network parameters including the number of TCP-streams in order to fully utilize the available bandwidth. DiskRouter can also use sophisticated routing to achieve better performance.

After letting Stork use the alternative protocol (in this case GridFTP) to perform the transfers for a while, we restarted the DiskRouter server at the SDSC site. This time, Stork immediately switched back to using DiskRouter for the transfers, since it was the preferred protocol of the user. Switching back to the faster protocol resulted in an increase in the performance. We repeated this a couple of more times, and observed that the system behaved in the same way every time.

This experiment shows that with alternate protocol fall-over capability, grid data placement jobs can make use of the new high performance protocols while they work and switch to more robust lower performance protocol when the high performance one fails.

5.2. Experiment 2: Testing the Run-time Protocol Auto-tuning. In the second experiment, we submitted another 500 data transfer requests to the Stork server. Each request was to transfer a 1.1GB image file (total 550 GB) using GridFTP as the primary protocol. We used third-party globus-url-copy transfers without any tuning and without changing any of the default parameters.

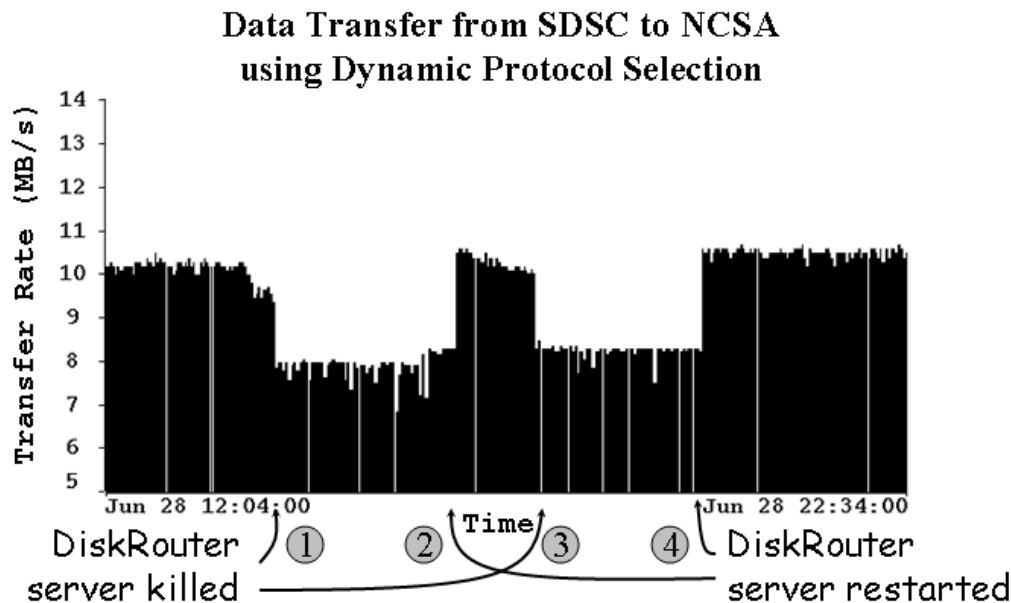


FIG. 5.2. *Dynamic Protocol Selection.* The DiskRouter server running on the SDSC machine gets killed twice at points (1) and (3), and it gets restarted at points (2) and (4). In both cases, Stork employed next available protocol (GridFTP in this case) to complete the transfers.

TABLE 5.1
Network parameters for gridFTP before and after auto-tuning feature of Stork being turned on.

Parameter	Before auto-tuning	After auto-tuning
parallelism	1 TCP stream	4 TCP streams
block size	1 MB	1 MB
tcp buffer size	64 KB	256 KB

We turned off the auto-tuning feature of Stork at the beginning of the experiment intentionally. The average data transfer rate that globus-url-copy could get without any tuning was only 0.5 MB/s. The default network parameters used by globus-url-copy are shown in Table 1. After a while, we turned on the auto-tuning feature of Stork. Stork first obtained the optimal values for I/O block size, TCP buffer size and the number of parallel TCP streams from the monitoring and tuning infrastructure. Then it applied these values to the subsequent transfers. Figure 5.3 shows the increase in the performance after the auto-tuning feature is turned on. We got a speedup of close to 20 times compared to transfers without tuning.

6. Future Work. We are planning to enhance the dynamic protocol selection feature of Stork, so that it will not only select any available protocol to perform the transfer, but it will select the best one. The requirements of ‘being the best protocol’ may vary from user to user. Some users may be interested in better performance, and others in better security or better reliability. Even the definition of ‘better performance’ may vary from user to user. We are looking into the semantics of how to define ‘the best’ according to each user’s requirements.

We are also planning to add a feature to Stork to dynamically select which route to use in the transfers and then dynamically deploy DiskRouters at the nodes on that route. This will enable us to use the optimal routes in the transfers, as well as optimal use of the available bandwidth throughout that route.

7. Conclusion. In this paper, we have shown a method to dynamically adapt data placement jobs to the environment at the execution time. We have developed a set of disk and memory and network profiling, monitoring and tuning tools which can provide optimal values for I/O block size, TCP buffer size, and the number of TCP streams for data transfers. These values are generated dynamically and provided to the higher level data placement scheduler, which can use them in adapting the data transfers at run-time to existing

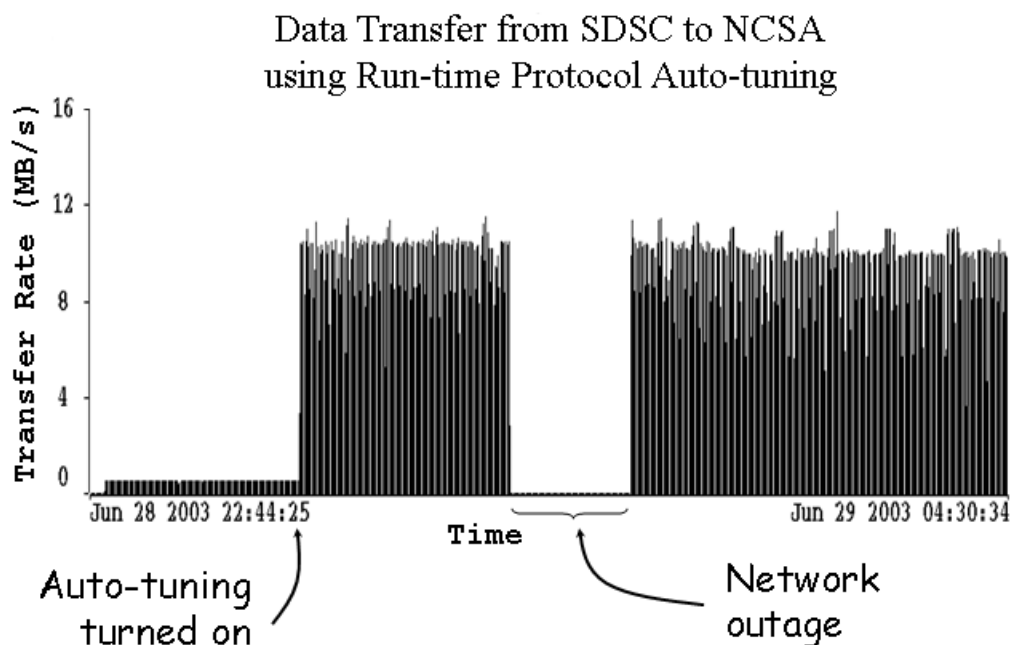


FIG. 5.3. *Run-time Protocol Auto-tuning.* Stork starts the transfers using the GridFTP protocol with auto-tuning turned off intentionally. Then we turn the auto-tuning on, and the performance increases drastically.

environmental conditions. We also have provided dynamic protocol selection and alternate protocol fall-back capabilities to provide superior performance and fault tolerance. With two experiments, we have shown that our method can be easily applied and it generates better performance results by dynamically switching to alternative protocols in case of a failure, and by dynamically auto-tuning protocol parameters at run-time.

Acknowledgements. We would like to thank Robert J. Brunner, Michelle Butler and Jason Alt from NCSA; Philip Papadopoulos, Mason J. Katz and George Kremenek from SDSC for the invaluable help in providing us access to their resources, support and feedback.

REFERENCES

- [1] B. ALLCOCK, J. BESTER, J. BRESNAHAN, A. CHERVENAK, I. FOSTER, C. KESSELMAN, S. MEDER, V. NEFEDOVA, D. QUESNEL AND S. TUECKE, *Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing*, in Proceedings of IEEE Mass Storage Conference", April 2001, San Diego, California.
- [2] M. BECK, T. MOORE, J. PLANK AND M. SWANY, *Logistical Networking*, Active Middleware Services, S. Hariri and C. Lee and C. Raghavendra, editors. Kluwer Academic Publishers, 2000.
- [3] J. BENT, V. VENKATARAMANI, N. LEROY, A. ROY, J. STANLEY, A. C. ARPACI-DUSSEAU, R. H. ARPACI-DUSSEAU AND M. LIVNY, *Flexibility, Manageability, and Performance in a Grid Storage Appliance*, in Proceedings of the Eleventh IEEE Symposium on High Performance Distributed Computing (HPDC11), July 2002, Edinburgh, Scotland.
- [4] F. BERMAN, R. WOLSKI, S. FIGUEIRA, J. SCHOPF AND G. SHAO, *Application Level Scheduling on Distributed Heterogeneous Networks*, in Proceedings of Supercomputing'96, Pittsburgh, Pennsylvania.
- [5] R. L. CARTER AND M. E. CROVELLA, *Dynamic Server Selection Using Bandwidth Probing in Wide-Area Networks*, Technical Report TR-96-007, Computer Science Department, Boston University, 1996.
- [6] CONDOR, *The Directed Acyclic Graph Manager*, <http://www.cs.wisc.edu/condor/dagman>, 2003.
- [7] C. DOVROLIS, P. RAMANATHAN AND D. MOORE, *What do packet dispersion techniques measure?*, in Proceedings of INFO-COMM, 2001.
- [8] M. FAERMAN, A. SU, R. WOLSKI AND F. BERMAN, *Adaptive Performance Prediction for Distributed Data-Intensive Applications*, in Proceedings of the IEE/ACM Conference on High Performance Networking and Computing, November 1999, Portland, Oregon.
- [9] M. FISK AND W. WENG, *Dynamic Right-Sizing in TCP*, in Proceedings of ICCCN, 2001.
- [10] I. FOSTER, C. KESSELMAN AND S. TUECKE, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, International Journal of Supercomputing Applications, 2001.

- [11] D. KOESTER, em Demonstrating the TeraGrid - A Distributed Supercomputer Machine Room, The Edge, The MITRE Advanced Technology Newsletter, (2) 2002.
- [12] G. KOLA AND M. LIVNY, *DiskRouter: A Flexible Infrastructure for High Performance Large Scale Data Transfers*, Technical Report CS-TR-2003-1484, University of Wisconsin, Computer Sciences Department, 2003.
- [13] T. KOSAR AND M. LIVNY, *Scheduling Data Placement Activities in the Grid*, Technical Report CS-TR-2003-1483, University of Wisconsin, Computer Sciences Department, 2003.
- [14] M. J. LITZKOW, M. LIVNY AND M. W. MUTKA, *Condor - A Hunter of Idle Workstations*, in Proceedings of the 8th International Conference of Distributed Computing Systems, (1988), pp. 104–111.
- [15] NLANR/DAST, *Auto Tuning Enabled FTP Client And Server: Autobuf*, <http://dast.nlanr.net/Projects/Autobuf>, 2003.
- [16] NLANR/DAST, *Iperf: The TCP/UDP Bandwidth Measurement Tool*, <http://dast.nlanr.net/Projects/Iperf/>, 2003.
- [17] S. OGURA, H. NAKADA AND S. MATSUOKA, *Evaluation of the inter-cluster data transfer on Grid environment*, in Proceedings of the Third IEEE/ACM Symposium on Cluster Computing and the Grid (CCGrid), May 2003, Tokyo, Japan.
- [18] R. RAMAN, M. LIVNY AND M. SOLOMON, *Matchmaking: Distributed Resource Management for High Throughput Computing*, in Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC7), July 1998, Chicago, Illinois.
- [19] B. SAGAL, *Grid Computing: The European DataGrid Project*, in Proceedings of IEEE Nuclear Science Symposium and Medical Imaging Conference, October 2000, Lyon, France.
- [20] J. SEMKE, J. MAHDAVI AND M. MATHIS, *Automatic TCP Buffer Tuning*, in Proceedings of SIGCOMM, pp. 315–323, 1998.
- [21] D. THAIN AND M. LIVNY, *The Ethernet Approach to Grid Computing*, in Proceedings of the Twelfth IEEE Symposium on High Performance Distributed Computing (HPDC12), June 2003, Seattle, Washington.
- [22] D. THAIN, J. BASNEY AND S. SON AND M. LIVNY, *The Kangaroo Approach to Data Movement on the Grid*, in Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC10), August 2001, San Francisco, California.
- [23] D. THAIN, T. TANNENBAUM AND M. LIVNY, *Condor and the Grid*, Grid Computing: Making the Global Infrastructure a Reality., Fran Berman and Geoffrey Fox and Tony Hey, editors. John Wiley and Sons Inc., 2002.
- [24] S. VAZHKUDAI, J. SCHOPF AND I. FOSTER, *Predicting the Performance of Wide Area Data Transfers*, in Proceedings of the 16th Int'l Parallel and Distributed Processing Symposium (IPDPS), 2002.
- [25] R. WOLSKI, *Dynamically Forecasting Network Performance to Support Dynamic Scheduling Using the Network Weather Service*, in Proceedings of the Sixth IEEE Symposium on High Performance Distributed Computing (HPDC6), August 1996, Portland, Oregon.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 9, 2003.

Accepted: September 1, 2003.



JUXMEM: AN ADAPTIVE SUPPORTIVE PLATFORM FOR DATA SHARING ON THE GRID

G. ANTONIU*, L. BOUGÉ†, AND M. JAN*

Abstract. We address the challenge of managing large amounts of numerical data within computing grids consisting of a federation of clusters. We claim that storing, accessing, updating and sharing such data should be considered by applications as an *external service*. We propose a *hierarchical architecture* for this service, based on a *peer-to-peer* approach. This architecture is illustrated through a software platform called JUXMEM (for Juxtaposed Memory), which provides transparent access to mutable data, while enhancing data persistence in a dynamic environment. Managing the *volatility of storage resources* is specially emphasized. As a proof of concept, we describe a prototype implementation on top of the JXTA peer-to-peer framework, and we report on a preliminary experimental evaluation.

Key words. data sharing, grid, peer-to-peer, hierarchical architecture, JXTA.

1. Introduction. A major contribution of the grid computing environments developed so far is to have decoupled *computation* from *deployment*. Deployment is then considered as an *external service* provided by the underlying infrastructure, outside the application. This service is in charge of locating and interacting with the physical resources, in order to efficiently schedule and map the computation. In contrast, as of today, no such sophisticated service exists regarding *data management* on the grid. Paradoxically enough, complex infrastructures are available for transparent computation scheduling on distributed sites, whereas the user is still left to explicitly store and transfer the data needed by the computation between these sites. At best, advanced FTP-like functionalities are proposed by existing environments. Within the context of a growing number of applications using large amounts of data, this *explicit data management* arises as a major limitation against the efficient use of modern computational grids.

Like deployment, we claim that an adequate approach to this problem consists in decoupling *data management* from *computation*, through an *external service* tailored to the requirements of scientific computation. In this work, we focus on the case of a grid consisting of a federation of distributed clusters. Such a *data sharing service* should meet the following two properties.

Persistence. The data sets used by the grid computing applications may be very large. Their transfer from one site to another may be costly (in terms of both bandwidth and latency), so such data movements should be carefully optimized. Therefore, a data management service should allow data to be stored on the grid infrastructure independently of the applications, in order to allow their reuse in an efficient way. Such a service should also provide data localization information, in order to co-operate with the computation scheduling service, and thereby enhance the global efficiency.

Transparency. Such a data management service should provide transparent access to data. It should handle data localization and transfer without any help from the programmer. Yet, it should make good use of additional information and hints provided by the programmer, if any. The service should also transparently use adequate replication strategies and consistency protocols to ensure data availability and consistency in a large-scale, dynamic architecture. In particular, it should support events such as computational and storage resources joining and leaving, or even unexpectedly failing.

At the same time, three main constraints need to be addressed:

Volatility and dynamicity. The clusters which make up the grid are not guaranteed to remain constantly available. Nodes may leave due to technical problems or because some resources become temporarily unavailable. This should obviously not result in disabling the data management service. Also, new nodes may dynamically join the physical infrastructure: the service should be able to dynamically take into account the additional resources they provide.

Scalability. The algorithms proposed for parallel computing have often been studied on small-scale configurations. Our target architecture is typically made of thousands of computing nodes, say tens of hundred-node clusters. It is well-known that designing low-level, explicit MPI programs is most difficult at such a scale. In contrast, high-level, peer-to-peer approaches have proved to remain effective at much larger scales.

*IRISA/INRIA Campus de Beaulieu, 35042 Rennes, FR. (Gabriel.Antoniu,Mathieu.Jan@irisa.fr).

†ENS Cachan/Bretagne Campus de Ker Lann, 35170 Bruz, FR. (Luc.Bouge@bretagne.ens-cachan.fr).

Mutable data. In our target applications, data are generally shared and can be modified by multiple partners. A large number of strategies have been proposed for handling data replication and data consistency, in the context of Distributed Shared Memory (DSM) systems. Again, these strategies and protocols have been designed with the assumption of a small-scale, static, homogeneous architecture, typically of clusters of few tens of nodes. A data sharing service for the grid should consider consistency protocols adapted to a dynamic, large-scale, heterogeneous architecture.

The type of service we propose is similar in some respects to several types of existing data management systems. However, these systems address only partially the goals and the three constraints mentioned above.

Non-transparent, large-scale data management. Currently, the most widely-used approach to data management for distributed grid computation relies on *explicit data transfers* between clients and computing servers. As an example, the Globus [7] platform provides data access mechanisms (Globus Access to Secondary Storage [3]) based on the GridFTP protocol [1]. Though this protocol provides authentication, parallel transfers, checkpoint/restart mechanisms, etc., it is still a FTP-like protocol which requires explicit data localization and transfer. Globus also integrates data catalogs, where multiple copies of the same data can be recorded. The management of these catalogs is manual: it is the user's responsibility to record these copies and make sure they are consistent: no consistency guarantee is provided by Globus.

Large-scale data storage. The IBP Project [2] provides a large-scale data storage system, consisting of a set of buffers distributed over Internet. The user can "rent" these storage areas and use them as temporary buffers for efficient data transfers across a wide-area network. IBP has been used by the Netsolve [18] computing environment to implement a service of persistent data. Transfer management is still at the user's charge. Besides, IBP does not handle dynamic join/departure of storage nodes and provides no consistency guarantee for multiple copies of the same data.

Transparent, small-scale data sharing. Distributed Shared Memory (DSM) systems provide transparent data sharing, via a unique address space accessible to physically distributed machines. Within this context, a variety of consistency models and protocols have been defined, in order to allow an efficient management of replicated data. These systems do offer transparent access to data: all nodes can read and write data in a uniform way, using a unique identifier or a virtual address. It is the responsibility of the DSM system to localize, transfer, replicate data, and guarantee their consistency according to some semantics. Nevertheless, existing DSM systems have generally shown satisfactory efficiency only on small-scale configurations, typically, a few tens of nodes [11].

Peer-to-peer sharing of immutable data. Recently, peer-to-peer (P2P) has proven to be an efficient approach for large-scale data sharing. The peer-to-peer model is complementary to the client-server model: the relations between machines are symmetrical, each node can be client in a transaction and server in another. This paradigm has been made popular by Napster [17], Gnutella [10], and now KaZaA [16]. We can note that these systems focus on sharing immutable files: the shared data are read-only and can be replicated at ease.

Peer-to-peer sharing of mutable data. Recently, some mechanisms for sharing mutable data in a peer-to-peer environment have been proposed by systems like OceanStore [8], Ivy [9] and P-Grid [6]. In OceanStore, for each data only a small set of primary replicas, called the *inner ring* agrees, serializes and applies updates. Updates are then multicast down a dissemination tree to all other cached copies of the data, called *secondary replicas*. However, OceanStore uses a versioning mechanism which has not proven to be efficient at large scales. Second, despite it provides hooks for managing the consistency of data, applications still have to use low-level mechanisms for each consistency model [12]. Third, published measurements on the performance of updates only assume a single writer per data block. Finally, servers making up inner rings are assumed to be highly available. The Ivy system has one main limitation: applications have to repair conflicting writes, thus the number of writers per data is very limited. Both Oceanstore and Ivy target general-purpose, persistent file storage, not data management for high-performance, computing grids where for example distributed matrices have to be moved using parallel transfers. P-Grid proposes a flooding-based algorithm for updating data, but assumes no conflicting writes. Besides, no experimental results have been published so far for this system.

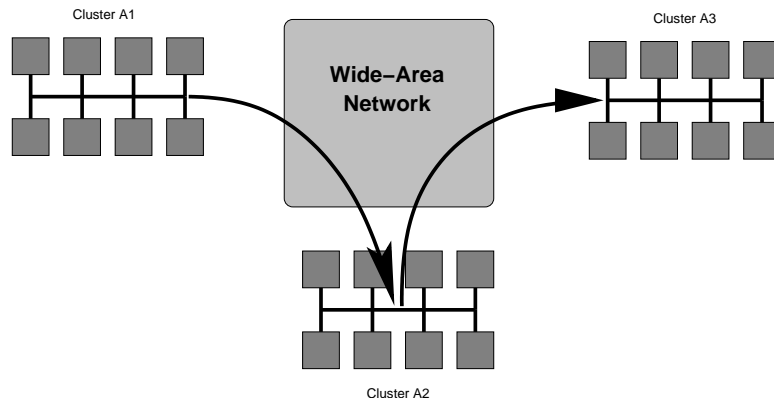


FIG. 2.1. Numerical simulation for weather forecast using a pipeline communication scheme with 3 clusters.

2. Designing a data sharing service for the grid.

2.1. Motivating scenarios. Let us consider a distributed federation of 3 clusters: A_1 , A_2 and A_3 , which co-operate together as shown on Figure 2.1. Each cluster is typically interconnected through a high-performance local-area network, whereas they are all coupled together through a regular wide-area network. Consider for instance a weather forecast simulation. Cluster A_1 may compute the forecast for a given day, then A_2 for the next day, and finally A_3 for the day after. Thus, A_3 uses data produced by A_2 , which in turn uses data produced by A_1 , as in a pipeline. Alternatively, cluster A_1 may simulate the weather forecast in a given country, while A_2 et A_3 simulate it for two neighboring countries.

Such simulations produce large amount of numerical data, and data-related actions are deeply intricated with computation. The data management systems described in the previous section do not provide any simple technique to support such designs. Consider for instance transferring data from A_1 to A_2 : a widely-used technique consists in *explicitly* writing the data on a disk within cluster A_1 , then use a file transfer tool to deposit them on a disk within cluster A_2 . The application is directly involved in this series of actions. In contrast, we propose to decouple the application from the data management, by making data storage and localization transparent with respect to the application. Cluster A_1 should only store the data within the federation-wide data management service, from which cluster A_2 could request them as needed. Data localization and transfer are then completely external to the applications.

Let us now suppose that our 3 applications no longer co-operate according to a pipeline scheme, but rather according to a *multiple-writers* scheme. For instance, each application simulates a single phenomenon part of the global weather forecast: say, wind, rain and clouds. In this case, each cluster needs data from the other ones in order to make progress. A data sharing service could allow the concurrent applications not only to read, but also to *write* to the globally shared data, while transparently handling data consistency. This is similar to DSM systems, but at a much larger scale, and in a fully dynamic context. Also, assume that some nodes fail in cluster A_2 . Some of the data necessary for A_3 could thus become unavailable. The data sharing service should also provide mechanisms to tolerate such faults, for instance, based on redundancy.

2.2. Design principles. We consider two major sources of inspiration for the design of a data sharing service for scientific grid computing:

DSM systems, which propose consistency models and protocols for efficient transparent management of *mutable data, on static, small-scaled configurations (tens of nodes)*;

P2P systems, which have proven adequate for the management of *immutable data on highly dynamic, large-scale configurations (millions of nodes)*.

These two classes of systems have been designed and studied in very different contexts. In DSM systems, the nodes are generally under the control of a single administration, and the resources are trusted. In contrast, P2P systems aggregate resources located at the edge of the Internet, with no trust guarantee, and loose control. Moreover these numerous resources are essentially heterogeneous in terms of processors, operating systems and network links, as opposed to DSM systems, where nodes are generally homogeneous. Finally, DSM systems are typically used to support complex numerical simulation applications, where data are accessed in parallel by

TABLE 2.1
A grid data sharing service as a compromise between DSM and P2P systems.

	DSM	Grid data service	P2P
Scale	10^1 – 10^2	10^3 – 10^4	10^5 – 10^6
Resource control and trust degree	High	Medium	Null
Dynamicity	Null	Medium	High
Resource homogeneity	Homogeneous (clusters)	Rather heterogeneous (clusters of clusters)	Heterogeneous (Internet)
Data type	Mutable	Mutable	Immutable
Application complexity	Complex	Complex	Simple
Typical applications	Scientific computation	Scientific computation and data storage	File sharing and storage

multiple nodes. In contrast, P2P systems generally serve as a support for storing and sharing immutable files. These antagonist features are summarized in the first and third columns of Table 2.1.

Our data sharing service targets physical architectures with features intermediate between DSM and P2P systems. We address scales of the order of thousands of nodes, organized as a federation of clusters, say tens of hundred-node clusters. At a global level, the resources are thus rather heterogeneous, while they can probably be considered as homogeneous within the individual clusters. The control degree and the trust degree are also intermediate, since the clusters may belong to different administrations, which set up agreements on the sharing protocol. Finally, we target numerical applications like heavy simulations, made by coupling individual codes. These simulations process large amounts of data, with significant requirements in terms of data storage and sharing. These intermediate features are illustrated in the second column of Table 2.1.

The contribution of this paper is namely to propose an architecture for such a data sharing service, which addresses the problem of managing *mutable data on dynamic, large-scale configurations*. Our approach aims at taking benefit of both DSM systems (transparent access to data, consistency protocols) and P2P systems (scalability, support for resource volatility and dynamicity).

2.3. The JXTA implementation framework. Our proposal is partly inspired by the P2P approach. It can usefully benefit from a platform providing basic mechanisms for peer-to-peer interaction. To our knowledge, the most advanced implementation platform in this area is JXTA [14]. The name JXTA stands for *juxtaposed*, in order to suggest the juxtaposition rather than the opposition of the P2P and client-server models. JXTA is a project originally initiated by Sun Microsystems.

JXTA is an open-source framework, which specifies a set of language- and platform-independent XML-based protocols [15]. JXTA provides a rich set of building blocks for the management of peer-to-peer systems: resource discovery, peer group management, peer-to-peer communication, etc.

Peers. The basic entity in JXTA is the *peer*. Peers are organized in networks. They are uniquely identified by IDs. An ID is a logical address independent of the location of the peer in the physical network. JXTA introduces several types of peers. The most relevant as far as we are concerned are the *edge peers* and *rendezvous peers*. Edge peers are able to communicate with other peers in the JXTA virtual network. They can also store advertisements of resources they discover in the network. Rendezvous peers have the extra ability of forwarding the requests they receive to other rendezvous peers. They can also offer a storage area for advertisements that have been published by edge peers. Finally, they are internally managed by JXTA using a *distributed hash table* (DHT) and are making up the frame of JXTA. They can thus be dynamically located in an efficient way. Joining, leaving, and even unexpected failing of rendezvous peers are supported by the JXTA protocols.

Peer groups. Peers can be members of one or several *peer groups*. A peer group is made up of several peers that share a common set of interests, e.g., peers that have the same access rights to some resources. The main motivation for creating peer groups is to build services collectively delivered by peer groups, instead of individual peers. Indeed, such services can then tolerate the loss of peers within the group, as its internal management is not visible to the clients.

Pipes. Communication between peers or peer groups within the JXTA virtual network is made by using *pipes*. Pipes are unidirectional, unreliable and asynchronous logical channels. JXTA offers two types of pipes: point-to-point pipes, and propagate pipes. Propagate pipes can be used to build a multicast layer at the virtual level.

Advertisements. Every resource in the JXTA network (peer, peer group, pipe, service, etc.) is described and published using *advertisements*. Advertisements are structured XML documents which are published within the network of rendezvous peers. To request a service, a client has first to *discover* a matching advertisement using specific localization protocols.

JXTA protocols. JXTA proposes six generic protocols. Out of these, two are particularly useful for building higher-level peer-to-peer services: the *Peer Discovery Protocol*, which allows for advertisement publishing and discovery; and the *Pipe Binding Protocol*, which dynamically establishes links between peers communicating on a given pipe.

The data sharing service that we propose is designed using the JXTA building blocks described above.

3. JUXMEM: a supportive platform for data sharing on the grid. The architecture of the data sharing service we propose, mirrors an architecture consisting of a federation of distributed clusters. The architecture is therefore *hierarchical*, and is illustrated through the proposition of a software platform called JUXMEM (for *Juxtaposed Memory*), whose goal is to be the foundation for a data sharing service for grid computing environments, like DIET [4].

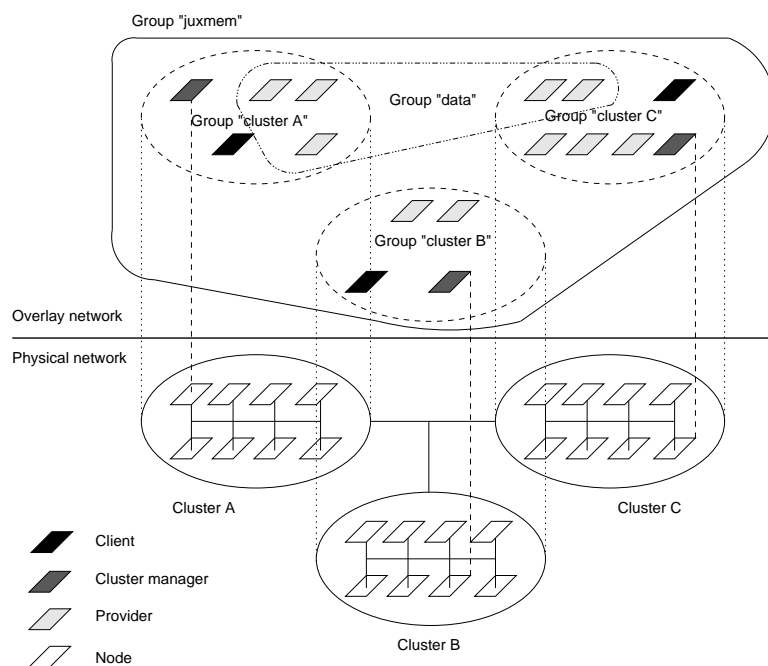


FIG. 3.1. Hierarchy of the entities in the network overlay defined by JUXMEM.

3.1. Hierarchical architecture. Figure 3.1 shows the hierarchy of the entities defined in the architecture of JUXMEM. This architecture is made up of a network of peer groups (*cluster* groups *A*, *B* and *C*), which generally correspond to clusters at the physical level. All the groups are inside a wider group which includes all the peers which run the service (the *juxmem* group). Each *cluster* group consists of a set of nodes which provide memory for data storage. We will call these nodes *providers*. In each *cluster* group, a node is in charge of managing the memory made available by the providers of the group. This node is called *cluster manager*. Finally, a node which simply uses the service to allocate and/or access data blocks is called *client*. It should be noted that a node can be at the same time a cluster manager, a client and a provider, but for the sake of clarity, each node plays only one role in the example illustrated on the Figure 3.1.

Each block of data stored in the system is associated to a group of peers called *data* group. This group consists of a set of providers that host copies of that data block. Note that a data group can be made up of

providers from different `cluster` groups. Indeed, a data can be spread over on several clusters (here A and C). For this reason, the `data` and `cluster` groups are at the same level of the group hierarchy. Note also that the `cluster` groups could also correspond to subsets of the same physical cluster.

Another important feature is that the architecture of JUXMEM is dynamic, since `cluster` and `data` groups can be created at run time. For instance, for each block of data inserted into the system, a `data` group is automatically instantiated.

API of the data sharing service. The Application Programming Interface (API) provided by JUXMEM illustrates the functionalities of a data sharing service providing data persistence as well as transparency with respect to data localization.

`alloc(size, attributes)` allows to create a memory area of the specified `size` on a cluster. The `attributes` parameter allows to specify the level of redundancy and the default protocol used to manage the consistency of the copies of the corresponding data block. This function returns an ID which can be seen at the application level as a data block ID.

`map(id, attributes)` allows to retrieve the advertisement of a data communication channel which has to be used to manipulate the data block identified by `id`. The `attributes` argument allows to specify parameters for the view of the data block desired by the client, like for instance what we call the degree of consistency: some clients may have weaker consistency requirements than the one ensured by the default protocol used to manage the data block.

`put(id, value)` allows to modify the value of the data block identified by `id`. The new value is then `value`.

`get(id)` allows to get the current value of the data block identified by `id`.

`lock(id)` allows to lock the data block identified by `id`. A lock is implicitly associated to each data block.

Clients which access a shared data block need to synchronize using this lock.

`unlock(id)` allows to unlock the data identified by `id`.

`reconfigure(attributes)` allows to dynamically reconfigure a node. The `attributes` parameter allows to indicate if the node is going to act as a cluster manager and/or as a provider. If the node is going to act as a provider, the `attributes` parameter also allows to specify the amount of memory that the node provides to JUXMEM.

3.2. Managing memory resources.

Publishing and placement of resource advertisements. Memory resources are managed using *advertisements*. Each provider publishes the amount of memory it offers within the `cluster` group to which it belongs, by the means of a *provider advertisement*. The cluster manager of the group stores all such advertisements available in his group. He is also responsible for publishing the amount of memory available in the cluster by using a *cluster advertisement*. This advertisement lists the amounts of memory offered by providers of the associated `cluster` group. These cluster advertisements are published inside the `juxmem` group, so that they can then be used by all the clients in order to allocate memory.

Cluster managers are thus in charge of making the link between the `cluster` group and the `juxmem` group. They make up a network organized using a DHT at the level of the `juxmem` group level, in order to build the frame of the data sharing service. This frame is represented by the ring on the Figure 3.2. Each cluster manager G1 to G6 is responsible for a cluster, respectively A1 to A6, each of which is made up of five nodes. At the level of the `juxmem` group, the DHT works as follows. Each cluster advertisement contains a list which enumerates the amounts of memory available in the cluster. Each individual amount is separately used to generate an ID, by means of a hash function. This ID is then used to determine the cluster manager responsible for all advertisements having this amount of available memory in their list. This cluster manager is not the peer that stores the advertisement, it only knows the cluster manager which published it in the JUXMEM network. This placement of cluster advertisements allows clients to easily retrieve advertisements in order to allocate memory: any request for a given amount of memory is directed to the cluster manager responsible for that amount of memory, using the hash mechanisms described above

Searching for advertisements is therefore short, and responses are exact and exhaustive, e.g., all the advertisements that include the requested memory size will be returned. But since using a DHT on memory sizes means to generate a different hash for each memory size, JUXMEM uses a parameterizable policy for the discretization of the space of memory sizes. Thus, JUXMEM will search for the minimum memory size, given by the policy used, that is superior to the one requested by clients. For example, if a client wants to allocate a memory area of 1280 bytes, JUXMEM will internally and automatically search for a memory area of 2048 bytes,

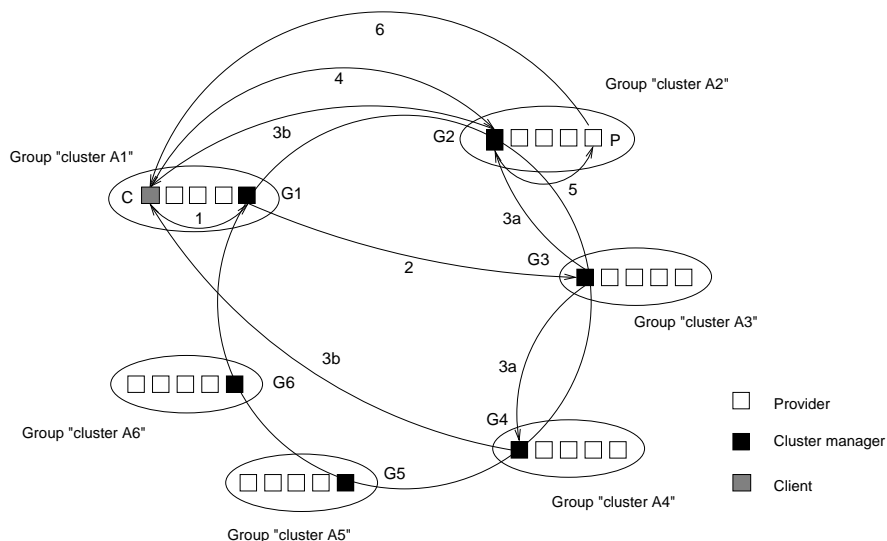


FIG. 3.2. Steps of an allocation request made by a client.

if it uses a power of 2 law for the space discretization. Providers also internally use the same law when offering memory areas, but provide the maximum memory size, given by the policy used, that is inferior to the one they wish to offer.

One of the constraints we fixed is to support the volatility of nodes which make up the clusters. Therefore, the advertisements published at a time t_1 can be invalid at the time $t_2 > t_1$, since providers can disappear from JUXMEM at any time. The mechanism used to manage this volatility of peers is based on republishing the cluster advertisements whenever a changing of the amount of memory provided is detected. Besides, advertisements have a limited but parameterizable lifetime, so it is necessary to periodically republish them.

Processing an allocation request. Clients make allocation requests by specifying the size of the memory area they want to allocate. The different steps for such a request, numbered on the Figure 3.2, are the following:

1. The client C of the `cluster` group A_1 wants to allocate a memory area of 8 MB with a redundancy degree of two. Consequently, it submits its request to the cluster manager G_1 to which it is connected.
2. The cluster manager G_1 then determines that the peer responsible of advertisements having a memory size of 8 MB in their list is the cluster manager G_3 , using the hash mechanism described previously. Therefore, the cluster manager peer G_1 forwards the request to G_3 .
3. The cluster manager G_3 then determines that cluster managers G_2 and G_4 match the criterion of the client, and asks them to forward their `cluster` advertisement to the client C .
4. The client C then chooses the cluster manager G_2 as the peer having the "best" advertisement: for instance the corresponding cluster offers a higher degree of redundancy than the cluster handled by the cluster manager G_4 . Thus, it submits its allocation request to G_2 .
5. The cluster manager G_2 receives the allocation request and handles it. If it can satisfy the request then it asks one of its providers, for example P , to allocate a 8 MB memory area. If the request cannot be satisfied, an error message is sent back to the client.
6. If the provider P can satisfy this request, it creates a 10 MB memory area, then sends back the advertisement of this memory area to the client C . P becomes the cluster manager of the associated `data` group, which means that it is responsible for replicating the data block stored in that memory area. If the provider P cannot satisfy the request, an error message is sent back to the cluster manager G_2 , which can try other provider peers of the `cluster` group.

If no providers can be found on the last step of an allocation request, an error message is sent back to the client. Then the client can restart the allocation request from step 4, e.g., with another cluster manager matching the requested memory size. Finally, if no cluster manager can allocate the memory area, the client increases the requested memory size and restarts the allocation request from the beginning. This can be done N times (for example $N = 3$) until the request is satisfied or an error is reported at the application level.

3.3. Managing shared data. When a memory area is allocated by a client, a `data` group is created on the chosen provider and an advertisement is sent to the client. This advertisement allows the client to communicate with the `data` group. This advertisement is published at the `juxmem`'s group level, but only the ID of this advertisement is returned at the application level. Access to data by other clients is then possible by using this ID: the platform *transparently* locates the corresponding data block.

Storage of data blocks is independent of clients. Indeed, when clients disconnect from JUXMEM, data blocks still remain stored in the data sharing service on the providers. Consequently, clients can have access to data blocks previously stored by other clients: they simply need to look for the advertisement of the `data` group associated with the data block (whose identifier is assumed to be known). The `map` primitive of the API of JUXMEM does this by taking in input the ID of the data block. In this way, the storage of data blocks is persistent.

Each data block is replicated on a fixed, parameterizable number of providers for a better availability. This redundancy degree is specified as an attribute at allocation time. The consistency of the different copies must then be handled. In this first version of JUXMEM, the use of a multicast at the level of the `juxmem` group solves this problem: the different copies of a same data block are simultaneously updated whenever a writing access is made. Alternative consistency models and protocols will be experimented in further versions. Note that clients which have previously read a data block are not notified of this update: clients do not store a copy of data block. Therefore, the result of a reading which is valid at a time t_1 , may not be valid at time $t_2 > t_1$. It is worth noting that this difference between client and providers allows to handle a high number of clients without having to deal with a high number of copies of data blocks. Synchronization between clients which concurrently access a data block is handled using the `lock/unlock` primitives.

3.4. Handling volatile providers. In order to tolerate the volatility of peers, a static replication of data on a fixed and parameterizable number of providers is not enough. Indeed, the set of providers hosting a copy of the same data block can successively become unavailable. A dynamic monitoring of the number of copies for data is therefore needed. Consequently, each `data` group has a manager (noted *data manager*) which is in charge of monitoring the level of redundancy of the data block. If this number goes below the one specified by clients, the data manager must search and ask a provider to host an extra copy of the data block. When the data manager decides to replicate it, it must first lock it (internally) in order to maintain consistency. The provider which will host this new copy is then responsible for unlocking it. A *timeout* mechanism followed by a *ping* test is used in order to detect if the provider became unavailable just before unlocking the data block. If it is the case, then the data manager unlocks itself the data block.

3.5. Handling volatile managers. If a cluster manager goes down, this could lead to the unavailability of resources provided by a whole cluster. The role of cluster manager (noted *main cluster manager*) is therefore automatically duplicated on another provider of the cluster (called *secondary cluster manager*). Managers periodically synchronize using a mechanism based on the exchange of provider advertisements, in order to find out new advertisements published. They can thus both know in a nearly accurate manner the amount of memory available in the cluster. A mechanism based on periodical heartbeats allows to dynamically ensure this duplication of cluster managers. Such a mechanism is also used for the data managers (see Section 3.4). Note that, the possible changes of managers in the `cluster` and `data` groups, due to the unavailability of managers, are not seen outside these groups. The availability of clusters and of data blocks is thus maximized, whereas the perturbation on the client side is minimized.

4. Implementation and preliminary evaluations.

4.1. Implementation of JUXMEM within the JXTA framework. In order to build a prototype of the software architecture described in the previous section, we have used the JXTA generic peer-to-peer framework (see Section 2.3). Our JUXMEM prototype uses the reference Java binding of JXTA (which is today the only binding compatible with the JXTA 2.0 specification). JUXMEM is written in Java and includes about 50 classes (5000 code lines).

JXTA fully meets the needs of JUXMEM. Thus, managers of `data` and `cluster` groups are based on JXTA's *rendezvous peers*. Indeed, managers have to know if providers are still alive by using a ping test in order to manage a cluster or a block of data. This can only be done if providers have previously published their advertisements on managers, which need to extract the address of each provider. Moreover, only JXTA's rendezvous peers can forward requests inside the JXTA network; these peers correspond to the role of *main*

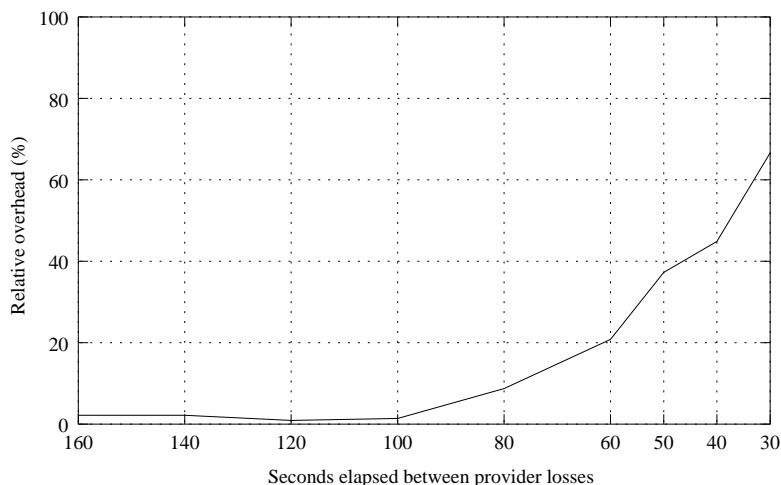


FIG. 4.1. Relative overhead due to the volatility of providers for a sequence `lock-put-unlock`, with respect to a stable system.

managers. For example, data managers have to forward access requests, made by clients, to providers hosting a copy of the data block. In the same way, cluster managers have to forward allocation requests, made by clients, to providers. Clients and providers which do not act as data managers for one or several blocks of data are based on JXTA's *edge peers*. Indeed, they do not have to play a role in the dynamic monitoring of the number of copies for a block of data in the system. Therefore, they do not have to store published provider advertisements. Moreover, clients only need to discover and store cluster advertisements which will allow them to allocate memory areas. The various groups defined in JUXMEM are implemented by JXTA's peer groups. The `juxmem` group implements a JXTA peer group service providing the API of JUXMEM (see Section 3.1). Finally, the communication channels of JXTA also offer the needed support for building multicast communications for simultaneously updating copies of the same block of data.

4.2. Preliminary evaluations. For our preliminary experiments, we used a cluster of 450 MHz Pentium II nodes with 256 MB RAM, interconnected by a 100 MB/s FastEthernet network.

We first measured the memory consumption overhead generated by the different JUXMEM peers with respect to the underlying JXTA peers used to build JUXMEM peers. This overhead is reasonable: it ranges between 5% and 7.4%.

We then measured the influence of the volatility degree of provider peers on the duration of a sequence `lock-put-unlock` executed in a loop by a client. This sequence in the loop is made on a data block stored in JUXMEM. The goal of this measure is to evaluate the relative overhead generated by the replications which take place in order to maintain a given redundancy degree for a given block of data. These replications are transparently triggered when the service detects that a provider holding a data block goes down. If these replications take place while a client accesses the data block being replicated, these accesses slow down.

The test program first allocates a small memory area (1 byte) on a provider belonging to cluster and writes to it a data block. The redundancy degree is set to 3. The allocation takes place on a cluster initially consisting of 16 providers and one cluster manager. 16 machines of the cluster previously described host a provider, one machine of the same cluster hosts a cluster manager and another machine of the same cluster hosts a client. The client executes a 100 iteration loop, and each iteration consists of a sequence `lock-put-unlock`.

During the execution of this loop, a random provider hosting a copy of the data is killed every δ seconds, where δ is a parameter of the experiment. In order to measure only the overhead due to the volatility of providers, the data manager of the associated group is never killed.

Figure 4.1 shows the relative overhead measured, with respect to a stable system (i.e. where no provider goes down during the loop execution: $\delta = \infty$). When the data manager detects that providers holding a copy of the data block have gone down, it tries to replicate the block on other available providers, which are not already hosting a copy of the data block. To ensure the consistency of the data during its replication, clients are not allowed to modify it. Therefore, the system has to internally lock the data. As a result of this internal locking, the sequence `lock-put-unlock` is longer, since the client is blocked and has to wait for the lock to be set free.

The curve profile is explained by the number of times the system replicates the data on providers, in order to maintain the redundancy degree specified by the client (which is 3 for this test). For the whole duration of our test, the number of triggered replications is given in the Table 4.1 as a function of the δ parameter.

For highly volatile systems ($\delta < 80$ s), the number of replications triggered becomes higher than 2 and the relative overhead becomes significant. For $\delta = 30$ s, it reaches more than 65% (10 replications triggered). However, in a realistic situation, the node volatility on the architecture we consider is typically a lot weaker ($\delta \gg 80$ s). For such values, the reconfiguration overhead is less than 5%. We can reasonably say that the JUXMEM platform includes a mechanism which allows to *dynamically* maintain a certain redundancy degree for data blocks, in order to improve data availability, *without significant overhead*, while authorizing node failures.

TABLE 4.1
Number of triggered replications when the volatility of provider peers evolves from 160 to 30 seconds.

Seconds	160	140	120	100	80	60	50	40	30
Number of triggered replications	1	1	1	1	2	2.5	5	5.5	10

5. Conclusion. This paper defines a *hierarchical* architecture for a data sharing service managing mutable data within a grid consisting of a federation of clusters. This architecture has been designed using a peer-to-peer approach, and demonstrated through the JUXMEM platform. Not only the architecture allows to reduce the number of messages to search for a piece of data, thanks to a hierarchical search scheme, but it also allows to take advantage of specific features of the underlying physical architecture. The management policy for each cluster can be specific to its configuration, for instance in terms of network links to be used. Thus, some clusters could use high-bandwidth, low-latency networks for intra-cluster communication, if available.

The JUXMEM user can allocate memory areas in the system, by specifying an area size and some attributes, such as a redundancy degree. The allocation primitive returns an ID which identifies the block of data. Then, data localization and transfer is fully transparent, since this ID is sufficient in order to access and manipulate the corresponding data wherever it is: no IP address nor port number needs to be specified at the application level.

Our architecture supports the volatility of all types of peers. This kind of volatility is also supported in peer-to-peer systems such as Gnutella or KaZaA, which enhance data availability thanks to redundancy. However, this is a side effect of the user actions. In contrast, our system *actively* takes into account this volatility: this allows not only to maintain a certain degree of data redundancy (as in systems like Ivy or CFS [5]), but also to support the volatility of peers with “specific” responsibilities (e.g., cluster managers, or data managers).

The implementation of a JXTA-based prototype has shown the feasibility of such a system. However, note that the design of JUXMEM is not dependent on JXTA. Actually, other libraries could be used, such as JavaGroups [13]. We used the Java version of JXTA, since this is the most advanced binding of JXTA, the only one compatible with the JXTA 2.0 specification.

The modular architecture of JXTA allows to easily add and remove services and/or protocols, including communication protocols. This should eventually allow the platform to take advantage of high-performance networks (such as Myrinet or SCI) for data transfer. We plan to address this problem in the future. We also plan to use JUXMEM as an experimental platform for different data consistency strategies supporting peer volatility, in order to build a configurable, adaptive data sharing service for mutable data. The final goal is to integrate this service into large-scale computing environments, such as DIET [4], developed at ENS Lyon. This will allow an extensive evaluation of the service, with realistic codes, using various data access schemes.

REFERENCES

- [1] B. ALLCOCK, J. BESTER, J. BRESNAHAN, A. CHERVENAK, L. LIMING, S. MEDER AND S. TUECKE, *GridFTP Protocol Specification*, GGF GridFTP Working Group Document, Sept. 2002.
- [2] A. BASSI, M. BECK, G. FAGG, T. MOORE, J. PLANK, M. SWANY AND R. WOLSKI, *The Internet Backplane Protocol: A study in resource sharing*, In 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2002), pages 194–201, Berlin, Germany, May 2002. IEEE.
- [3] J. BESTER, I. FOSTER, C. KESSELMAN, J. TEDESCO AND S. TUECKE, *GASS: A data movement and access service for wide area computing systems*, In 6th Workshop on I/O in Parallel and Distributed Systems (IOPADS '99), pages 77–88, Atlanta, GA, May 1999. ACM Press.

- [4] E. CARON, F. DESPREZ, F. LOMBARD, J.-M. NICOD, M. QUINSON AND F. SUTER, *A scalable approach to network enabled servers*, In B. Monien and R. Feldmann, editors, 8th International Euro-Par Conference, volume 2400 of Lecture Notes in Computer Science, pages 907–910, Paderborn, Germany, Aug. 2002. Springer-Verlag.
- [5] F. DABEK, F. KAASHOEK, D. KARGER, R. MORRIS AND I. STOICA, *Wide-area cooperative storage with CFS*, In 18th ACM Symposium on Operating Systems Principles (SOSP '01), pages 202–215, Chateau Lake Louise, Banff, Alberta, Canada, Oct. 2001.
- [6] A. DATTA, M. HAUSWIRTH AND K. ABERER, *Updates in highly unreliable, replicated peer-to-peer systems*, In 23rd International Conference on Distributed Computing Systems (ICDCS 2003), pages 76–87, Providence, Rhode Island, USA, May 2003.
- [7] I. FOSTER AND C. KESSELMAN, *Globus: A metacomputing infrastructure toolkit*, The International Journal of Supercomputer Applications and High Performance Computing, 11(2):115–128, 1997.
- [8] J. KUBIATOWICZ, D. BINDEL, Y. CHEN, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, W. WEIMER, C. WELLS AND B. ZHAO, *OceanStore: An architecture for global-scale persistent storage*, In 9th International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS 2000), number 2218 in Lecture Notes in Computer Science, pages 190–201, Cambridge, MA, Nov. 2000. Springer.
- [9] A. MUTHITACHAROEN, R. MORRIS, T. M. GIL AND B. CHEN, *Ivy: A read/write peer-to-peer file system*, In 5th Symposium on Operating Systems Design and Implementation (OSDI '02), Boston, MA, Dec. 2002.
- [10] A. ORAM, *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, chapter Gnutella, pages 94–122, O'Reilly, May 2001.
- [11] J. PROTIĆ, M. TOMASEVIĆ AND V. MILUTINOVIĆ, *Distributed Shared Memory: Concepts and Systems*, IEEE, Aug. 1997.
- [12] S. RHEA, P. EATON, D. GEELS, H. WEATHERSPOON, B. ZHAO AND J. KUBIATOWICZ, *Pond: the oceanstore prototype*, In 2nd USENIX Conference on File and Storage Technologies (FAST '03), California, CA, USA, Mar. 2003.
- [13] JAVAGROUPS, <http://www.javagroups.com/javagroupsnew/docs/index.html>
- [14] THE JXTA PROJECT, <http://www.jxta.org/>
- [15] JXTA v2.0 PROTOCOL SPECIFICATION, <http://spec.jxta.org/nonav/v1.0/docbook/JXTAProtocols.pdf>, Mar. 2003.
- [16] KAZAA, <http://www.kazaa.com/>
- [17] NAPSTER PROTOCOL SPECIFICATION, <http://opennap.sourceforge.net/napster.txt>, Mar. 2001.
- [18] THE NETSOLVE PROJECT, <http://icl.cs.utk.edu/netsolve/>

Edited by: Wilson Rivera, Jaime Seguel.

Received: June 26, 2003.

Accepted: September 1, 2003.



PROGRESSIVE RETRIEVAL AND HIERARCHICAL VISUALIZATION OF LARGE REMOTE DATA

HANS-CHRISTIAN HEGE*, ANDREI HUTANU* , RALF KÄHLER*, ANDRÉ MERZKY* , THOMAS RADKE†, EDWARD SEIDEL† , AND BRYGG ULLMER†

Abstract.

The size of data sets produced on remote supercomputer facilities frequently exceeds the processing capabilities of local visualization workstations. This phenomenon increasingly limits scientists when analyzing results of large-scale scientific simulations. That problem gets even more prominent in scientific collaborations, spanning large virtual organizations, working on common shared sets of data distributed in Grid environments. In the visualization community, this problem is addressed by distributing the visualization pipeline. In particular, early stages of the pipeline are executed on resources closer to the initial (remote) locations of the data sets.

This paper presents an efficient technique for placing the first two stages of the visualization pipeline (data access and data filter) onto remote resources. This is realized by exploiting the “extended retrieve” feature of GridFTP for flexible, high performance access to very large HDF5 files. We reduce the number of network transactions for filtering operations by utilizing a server side data processing plugin, and hence reduce latency overhead compared to GridFTP partial file access. The paper further describes the application of hierarchical rendering techniques on remote uniform data sets, which make use of the remote data filtering stage.

1. Introduction. The amount of data produced by numerical simulations on supercomputing facilities continues to increase rapidly in parallel with the increasing compute power, main memory, storage space, and I/O transfer rates available to researchers. These developments in supercomputing have been observed to exceed the growth of commodity network bandwidth and visualization workstation memory/performance by a factor of 4 [11]. Hence, it is increasingly critical to use remote data access techniques for analyzing this data. Among other factors, this tendency is strengthened by the increasing prominence of large, spatially distributed scientific collaborations working on common, shared sets of data. Under these conditions, the simple approach of (partial) data replication for local data analysis does not scale.

The sheer size of existing data sets creates a demand for flexible and adaptive visualization techniques, such as hierarchical rendering or viewpoint dependent resolution. Such techniques can reduce the initial amount of data to be visualized by maintaining the overall visual impression of the full data set. This can be achieved (e.g.) by retrieving the portions of the data set which are important to the user; or by retrieving low resolution versions of the full data set first, and refining this data later. Remote access to partial *interesting* portions of large data files can significantly support these techniques.

One major problem of naive remote data access techniques is the inherent difficulty in handling meta data for large data sets. Meta data is the highly structured set of information describing the data set, containing (e.g.) the number of samples per coordinate axis and the data volume bounds within physical space. While the metadata itself is relatively small, meta data access is often connected with many small read operations and many seek operations. However, individually requesting many seeks over a remote, potentially high-latency connection is quite inefficient for protocols that do not support transactions over higher level operations [13, 19].

In general, these developments ultimately require distributing the pipeline used for data visualization. The present paper describes techniques useable for distributing early stages of this visualization pipeline. Specifically, we enable the application to efficiently access portions of remote large data sets present in the HDF5 file format [2]. This general approach can be adapted both to other file formats and other access patterns. The paper further presents higher level visualization techniques which utilize these data access mechanisms to provide adaptive and progressive rendering capabilities.

The paper is structured as follows. First, we describe the problem space our approach is targeting in more detail in sect. 2. Next, we relate our research to other relevant research activities (see sect. 3). In sect. 4 follows an overall description of the techniques we developed. Sect. 5 and 6 describe the main components in more technical detail. The paper concludes with two sections about our results and an outlook for future work.

2. Scenario. The increasing gap between resources available at remote supercomputing centers and on the local workstations of individual researchers is one of the major motivations for our research. In particular, we aim to improve the access to Grand Challenge simulation results as produced by numerous research collaborations

*Zuse Institute Berlin (ZIB), <http://www.zib.de/>, {hege, hutanu, kaehler, merzky, ullmer}@zib.de

†Albert Einstein Institute (AEI), <http://www.aei.mpg.de/>, {radke, seidel}@aei.mpg.de

around the world [12, 25, 26]. These simulations tend to drive the resource utilization of supercomputer resources to the available maximum, and often produce immense amounts of data during single simulation runs.

As an exemplary application we consider numerical relativity simulations performed in the Cactus simulation framework [10]. Among other things, this framework provides the simulation code with an efficient I/O infrastructure to write data to HDF5 files. The astrophysical simulations in question write data for scalar, vector and tensor fields (as components stored in separate data sets or files), and parameters for simulation runs. A typical size for a data file is on the order of tens of gigabytes¹.

Visualization of this data during post simulation analysis usually does not require access to the complete data set. For typical production runs, where many different physical fields are written to disk, only a couple of these fields are visualized later. The data sets and subsets that are to be visualized are not initially known, but depend on interactive selections by the user (timestep, field, resolution, spatial area, etc.). For our target users, this flexibility needs to be maintained as far as possible.

Within these constraints, our target scenario is the following:

A scientist performs a large scale simulation run, utilizing one or more supercomputing resources at different locations. The simulation run produces up to TBytes of data, by storing various scalar and vector fields to HDF5 files. These HDF5 files are created according to a custom predefined structure.

After the simulation finishes, members of the scientists' collaboration wish to visualize the data, or portions hereof, from remote workstations. They would like to use standard visualization techniques from their visualization environment. They also wish to interactively choose the data fields to be visualized, and to interactively change the spatial selection and resolution for the data.

Ideally, the data transfer and visualization are adaptive to the available network connectivity, and hides data distribution details from the user.

This scenario defines the problem space we are targeting. We explicitly do not expect to find data on the remote systems which are, by pre- or postprocessing, specifically prepared for later visualization. We also want to provide a solution for environments with notorious short supply of I/O bandwidth and compute resources. And we want to enable remote visualization for a broad width of end users, connected to the Grid by a wide range of network types and with varying, potentially low end commodity systems. The ability of the visualization pipeline to be *adaptive* to that range of boundary conditions is a central point of our efforts—the focus of the paper on progressive data retrieval patterns and on hierarchical rendering techniques emphasizes this.

3. Related Work. To support the scenario we presented, it is ultimately necessary to distribute the pipeline used for data visualization. In principle, there are many possible ways to distribute this pipeline (fig. 3.1) over remote resources. The distribution schemes used in real world systems are limited by the communication requirements for transferring data between the stages of the pipeline, and by the complexity of the resulting distributed software systems.

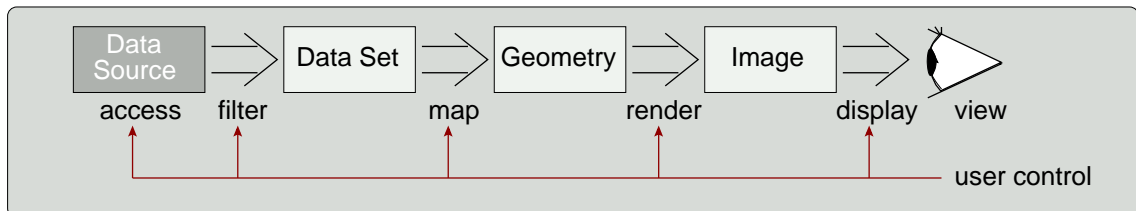


FIG. 3.1. Most visualization systems share the same underlying visualization pipeline [27]. The components of the pipeline can be freely distributed, in principle, as the communication elements between these components have different demands on latency and bandwidth required. All elements of the pipeline should be controlled by the end user or by the application.

Early stages of the pipeline—remote access and remote filtering—potentially need to transfer and process large amounts of data, but show considerable flexibility with respect to latency. Also, by distributing these early

¹With a spatial resolution of 256 cubed, this corresponds to only a few scalar fields and one vector field in 64 bit, for 1000 time steps of evolution, with every 10th step saved to disk.

stages, it is possible to completely hide the data locality from application and end user. Remote access solutions as NFS [4] and AFS [3] allow transparent utilization of standard (local) file I/O techniques. However, systems like NFS and AFS are problematic in the administrative maintenance. For widely distributed environments spanning multiple administrative domains these solutions are not applicable.

Common remote data access techniques crossing administrative boundaries are marked by several limitations. Some, like SCP and FTP, do not support access to partial files, which is not acceptable for our purpose of adaptive visualizing. Other techniques fail to deliver the performance required for interactive data visualization. For example, the GridFTP support for access to remote files with the partial file access feature [9] is inefficient for meta data access. Due to the file format chosen by HDF5, meta data is not necessarily stored in a continuous file space, but instead scattered in a hierarchical binary tree. Also, a single read on the HDF5 API level may be translated by the library into many individual low-level seek/read operations on the virtual file driver level. Other protocols are similarly lacking in support for transactions of higher level operations [13, 19].

Remote filtering techniques often integrate models of meta data and data structures, and can perform the data access efficiently². Also, putting the remote filter on the remote site can significantly reduce the amount of data to be transferred over the net, and ensures that only the data actually needed for the visualization process is retrieved and transferred. A standard problem for remote filtering is that this process needs to integrate a model of the data structures it is operating upon. It is difficult or impossible to implement filtering without explicit information about what is to be filtered, and this information is difficult to express in a general way that is applicable over a broad range of data formats and models. Hence, remote filtering techniques are often limited to specific file and data types, and to specific filtering operations.

The Data Cutter project [14] is another well known representative of the remote filtering approach. It provides the application programmer with a flexible and extensible filter pipeline to access portions of the original data set. Compared to our approach, there are several main differences. First, the data cutter *requires* the data to be stored in chunked data files in order to benefit from its boundary box indexing scheme, since all chunks with a bounding box at least partly overlapping with the area of interest are completely read into memory, and passed to the filter pipeline. Also, since all filters pass data using network communication, the total network load is much higher than for our approach, where the filter resides at the data source, and is tightly coupled to the data access stage. Further, our utilization of standard Grid tools (GridFTP and GSI) seems more appropriate for the targeted Grid environment. On the other hand, Data Cutters user definable filter pipeline is more flexible than our approach.

One widely used compromise for remote filtering is the usage of preprocessed data sets: during the simulations I/O stage or during a post processing step, filter operations are applied to create new data sets on the remote resources. These data sets are stored in optimized form making later remote access and visualization very efficient. In the future, more and more simulation frameworks will support such features, not at least in order to improve their own I/O characteristics, i.e. due to compression on the fly, but also to enable the efficient handling of the very large data sets, after completion of the source simulation. Wavelet transformed data storage is an excellent example of that technique [22], which allows lossless compression, and adaptive, efficient offline access to optimally resolved data samples. Other example filters create octrees [18] or similar structured representations [21], or provide progressive mesh generation.

For the problem space we described with our scenario, pre applied filters are no valid option, since they either need to be integrated into the simulation I/O code, what they aren't in our case; or they need to be executed via external jobs on the remote resource. This duplicates the storage needed and potentially performs excess work, thereby wasting costly supercomputing resources.

After filtering, visualization algorithms work on the data and map essential features into geometries (including color and texture information, etc.). The next stage renders images from these geometrical representation. In the future, these stages may also be executed close to the data source, on the supercomputer itself. This would be the most efficient way to handle large simulation data, since the amount of data to be transferred during the later stages of the visualization pipeline typically decreases significantly. Completely changed access patterns to remote data can significantly reduce the amount of data transferred. Visualization algorithms using such patterns [23], in particular for large data, are seen as use cases for the presented work.

The best prospects of deploying such scenarios have those environments containing PC-cluster based supercomputers. Here, adding commodity graphics boards to all nodes does not increase the total costs significantly,

²If the filter stage is located on the remote site, the data access is often performed local to the filter.

but allows high performance image rendering. These types of clusters are becoming increasingly common, but are still rare in the top500 [6]. For the collaborative and highly interactive visualization scenario we envision, the feedback to the remote and distributed rendering system gets important, and complex. Also, in perhaps the most important point, the field is currently missing sufficiently flexible software solutions which are able to realize such scenarios. Promising approaches do exist through work such as [8, 7, 24], and we expect major progress in that field over the next decade.

4. Architecture. Our proposed remote data access scheme builds upon the GridFTP protocol [9]. GridFTP is a Grid-aware extension to the standard FTP protocol. Amongst others, it provides a flexible server side processing feature, and allows specification of *custom* operations on remote data. These operations are performed by corresponding custom extensions (“plugins”) to the GridFTP server. This technique is described in more detail in sect. 5. We utilize these server side data processing capabilities to perform data filtering operations on the scientific data sets. As described, the data sets are stored remotely in HDF5 format. Our plugin to the GridFTP server accesses this data locally via the HDF5 library, and performs data filtering on the fly.

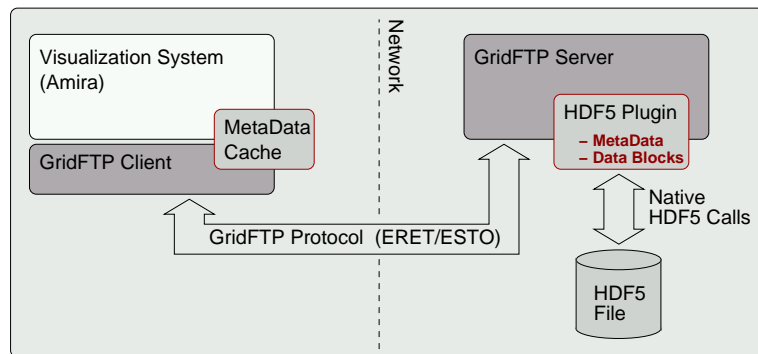


FIG. 4.1. The GridFTP protocol transports ERET commands from the visualization system to the GridFTP server, which forwards them to the HDF5 plugin. This way, the plugin can perform I/O operations plus filtering and data type conversion on the HDF5 file with full local performance. Data is transferred back via ESTO commands, and is written into the memory buffer of the visualization process.

An important element for the architectural decisions is the usage of the HDF5 file format [1]. Given the complexity of this format and the ongoing improvement efforts concerning the associated API, the decision was to use the existing API and to have the remote access procedures either on top of the API or as also described in sect. 3 underneath of it. The architecture described in this work has the remote operations on top of the HDF5 API, a limited set of high-level operations was chosen to be implemented by making use of the existing API, and these operations were integrated in the GridFTP server to be executed at the remote site.

A complete visualization session is performed as follows. The user selects a data file to be visualized by browsing the remote file space. Next, a connection to the remote GridFTP server is established, using the users GSI credential. The server plugin is utilized to perform an extraction of the files meta data (see sect. 5), which is then transferred to the visualization host and cached on the local file system. The visualization system accesses this local HDF5 file, extracts all needed information (number of time steps, bounding box, resolution, ...), and creates an octree hierarchy fitting the data set. The user can interactively specify the depth of the hierarchy. As the user then triggers various visualization operations on the data (to produce orthoslices, height fields, volumetric renderings), the octree blocks are scheduled in a separate thread for data reading. The read requests are served according to a priority tag defined for the visualization, and each trigger a GridFTP data access. This GridFTP data access utilizes our remote GridFTP server side data processing plugin. It extracts the data in the block specific resolution and returns this data. On arrival, the data is stored within the octree hierarchy, and the visualization is triggered to update the rendering by including the newly arrived data. On user request (e.g., next timestep) or timeout, all pending block reads can be canceled. Our visualization techniques (see sect. 6) use these features for dynamic data access to optimize visualization performance by requesting data blocks close to the viewpoint first, and by progressively improving data (and image) resolution.

5. GridFTP. As described in sect. 4, the GridFTP protocol plays a central role in our data access schema. GridFTP is mostly used for network file transfer, whereby this paper explores its usage for memory to memory

transfer. This approach gives us a number of advantages if compared to approaches implemented on top of custom or proprietary protocols.

1. GridFTP allows for server side data processing, which we utilize for data filtering.
2. The GridFTP protocol, as an extension to the standard FTP protocol, is well known and reliable.
3. It allows the incorporation of standard servers for solutions with limited functionality.³
4. The GridFTP infrastructure takes care of:
 - establishing the data connection;
 - ensuring authentication and authorization;
 - invoking the data filter plugin; and
 - performing the data transfer;

In this way, the data transfer task is reduced to filling a buffer on the writing and reading it on the receiving end.

The following subsections describe the server side processing in more detail, and specify the low level operations we use.

5.1. Server-Side Processing. As described before, the GridFTP protocol enables support for adding custom commands for server side data processing [9]. Specifically, the plugins offered by a server define sets of ERET and ESTO parameters that correspond to the data filter module implemented by the plugin⁴. The extended store (ESTO) and extended retrieve (ERET) commands of the GridFTP protocol are defined as following:

```
ESTO <module_name>=<modules_parms>" <filename>
ERET <module_name>=<modules_parms>" <filename>
```

`module_name` is a server-specific string representing the name of the module to be used. The second string (`module_parms`) is module specific and defines the operation to be performed by the module. The last parameter (`filename`) specifies the file to be processed, which can be any file that can be processed by the given module. In our case, any HDF5 file.

5.2. Operations. We use this ERET/ESTO mechanism to define two operations that can be applied to HDF5 files: one for meta data filtering, and a second one for data access.

Meta Data Filtering. The first operation is the filtering of meta data from the HDF5 file. This is achieved by creating a filtered copy of the original file. Toward this end, the module reads and parses the original file, and writes the meta data information to a copy of the file. However, when copying (writing) a data set, we use the HDF5 filter interface and apply a filter to the original files data set. This filter reduces all data sets to zero length⁵. Thus, the only resulting differences between the generated file and the original one are in the data array and storage layout of the data sets. All other information—e.g., the hierarchy (groups), attributes, and data set information (name, data type and data space)—is preserved. While this approach might seem like a significant overhead, it is in fact very fast, due to the good performance of HDF5.

The generated file is transferred to the requesting client using GridFTP. The ERET command for requesting the meta data file is:

```
ERET Hdf5="METADATA" <filename>
```

`filename` is the file from which the meta data will be extracted. Given the now dramatically reduced size of the file, the transfer time is very small relative to the transfer time of the original data⁶. After the high-level filtering call is executed remotely and the transfer is finished, the client can access the local meta data file using the standard HDF5 API. In this way, we avoid to execute each HDF5 API call remote, and still offer the user the flexibility of the original API for meta data access. Because the data set structures within this temporary local file do not contain actual data, the standard API cannot be used for data access. For this task, we provide a second API call.

³backward compatible with FTP, by using normal FTP we could transfer the file to a local disk cache; for standard GridFTP server(without plugins) we use direct partial file access (ERET PART, for filtering inefficient).

⁴Not all servers implement the same set of modules. In the current implementation, the plugins are compiled together with the server, and are statically linked.

⁵Actually, for technical reasons internal to HDF5 the length is 1.

⁶See sect. 7 for the times for meta data loading

Data Set Reading and Subsampling. The second operation performs data selection and filtering. By knowing the data set coordinates (dimensions, data type) from the now locally available meta data, the client can choose to read an entire data set, or a portion of the data set. The HDF5 data sets logically group the actual data within multidimensional arrays named "data spaces." The model we use to specify a portion from a data set is based on the HDF5 "hyperslab" model. A hyperslab describes either a contiguous collection of points, or a regular pattern of points or blocks in the data spaces. A hyperslab is specified by four parameters:

- origin: the starting location;
- size: the number of elements (or blocks) to select along each dimension;
- stride: the number of elements to separate each element (or block) to be selected; and
- block: the size of each block selected from the data set.

All of these parameters are one-dimensional lists, with lengths equal to the number of dimensions of the data set. The elements of these lists specify data array lengths or offsets for corresponding dimensions of the data arrays. Currently the size of element blocks is predefined to one, which is adequate for the targeted visualization scenario. In future work, we will extend the protocol to accept variable block sizes.

Our current mechanism for specifying the hyperslab coordinates takes the following form:

```
ERET Hdf5="BLOCK:NAME=<datasetname>;\  
      DIMENSIONS=<dims>;\  
      ORIGIN=<orig0>,<orig1>,...,<orign>;\  
      SIZE=<size0>,<size1>,...,<sizen>;\  
      SAMPLING=<sampling0>,<sampling1>,...,<samplingn>"  
      <filename>
```

`datasetname` is the fully qualified name (including the path to the data set) of the data set from which data should be read; `orig0` to `orign` are the coordinates of the first element to be selected from the data set; `size0` to `sizen` are the number of elements to be selected in each dimension; and `sampling0` to `samplingn` represent the distance between two selected elements for each dimension.

This request is sent to the server. The server opens the file *filename*, opens the given data set, and reads the portion of the file specified by the given parameters. This procedure is performed via native HDF5 library calls. Next, the retrieved data is sent via the GridFTP connection to the client, which will convert the data to the local byte order if needed. To determine if conversion is necessary, the first 32 bits sent by the server represent an integer with the value of 1, encoded using the servers byte ordering.

The approach we have taken in creating this limited HDF5 API wrapper does reduce the flexibility provided by the original API. Nonetheless, for our visualization scenario this API is appropriate, and makes significant steps toward maximizing overall performance. To retain the flexibility of the original API, one approach would be to execute each native API call remotely. In this case, the cost per call is at least that of the network latency. This, combined with the relatively large number of calls needed for example to gather the meta data from the file, significantly reduces the performance. This motivates the usage of higher level API wrappers, as the one we have implemented. However, such wrappers need not to be as limited as our current version of course.

5.3. Security. The security model used by the GridFTP server is GSI (Grid Security Infrastructure) [17]. The client needs to hold a valid GSI proxy containing a security credential with limited validity. The proxy represents a Distinguished Name (DN) that must be present in the `grid-mapfile` of the server machine in order for the server to accept the connection. This proxy is used to authenticate the client without using passwords. After the connection is established, the server front end starts the MPI-based back end. This back end runs under the local identity to which the DN is mapped. The back end is responsible for all subsequent operations, including the filtering operations. This ensures that only authorized clients can access the information from the original file.

6. Adaptive Visualization. We utilize the previously described techniques for data access and filtering to generate a level-of-detail representation of the remote data set in the visualization phase.

First, the meta data—i. e. information about the number of data samples per coordinate axis and the data volume extension in physical space—is retrieved (see sect. 5.2). With help of this information, and a selectable minimal resolution of the data, an octree structure is generated, which initially contains no data other than the parent-child relations and position and extensions of the tree nodes. The root node of the structure will store a coarse representation of the whole data volume. This is recursively refined by subnodes with higher spatial resolution until the resolution of the original data is reached.

Next, the data for the octree nodes is requested from the reader module, starting at the root node. The order in which nodes are refined is determined by the distance from a user-defined point-of-interest, which might be the camera position or an arbitrary point within the data volume. Subregions of the data sets closer to this point are requested with higher priority than those which are further away. The position and resolution parameters for each request are specified and sent to the remote machine as described in sect. 5.2.

The reader runs in a separate thread, so the visualization routines are not blocked during the loading phase. Each time a data block has arrived, the visualization module is notified, and this new data is reflected in the next rendered frame of the visualization.

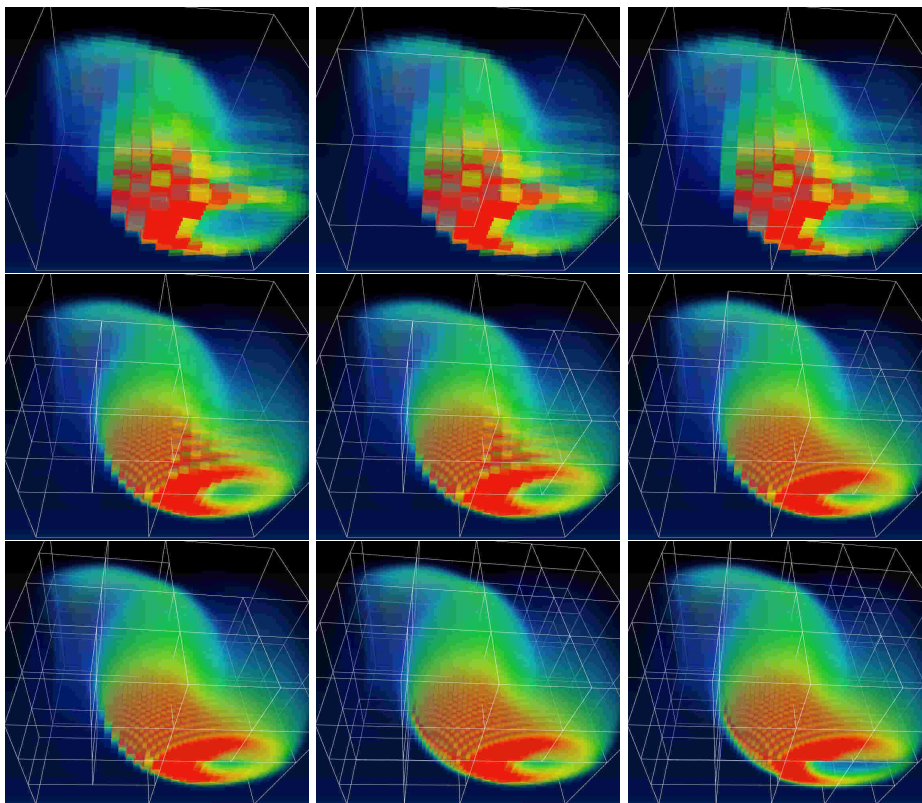


FIG. 6.1. The sequence depicts the volume rendering of a remote data set. First, a coarse resolution representation of the data is generated on-the-fly and transferred to the local visualization client. Next, subregions closer to the point-of-interest (in this case, the camera position) are requested and integrated at progressively higher resolutions.

Besides hierarchical visualization modules for orthoslicing and the display of height fields, we implemented a 3D texture-based volume rendering module for octrees. The octree is traversed in a view-consistent (back-to-front) order, starting at the root node. A node is rendered, if two criteria are fulfilled:

- The data for this node is already loaded (otherwise, the traversal of the associated subtree is stopped).
- The data for the subnodes is not loaded yet (otherwise, the node is skipped and the subnodes are visited).

Once a node is selected, it is rendered utilizing the standard approach for volume rendering with 3D textures, as proposed in (e.g.) [16, 15]. The 3D texture is sampled on slices perpendicular to the viewing direction and blended in the frame buffer.

In order to take advantage of the multi-resolution structure of the data for fast rendering, the sample distance of the slices is set with respect to the resolution level of the actual node, as proposed in [29].

7. Results.

7.1. Implementation. The implementation of the remote data access infrastructure we have described is based on an experimental version of the GridFTP server provided by the Globus Group. This server is not

part of the Globus software distribution as of yet. It supports the addition of compile-time plugins (written in C) for handling specific incarnations of the ERET/ESTO protocol commands. Although ERET and ESTO are specified in the GridFTP protocol version 1.0, there is currently no other implementation of this feature available other than the basic support for partial file access and striped data access. There are good prospects for this feature to be present in various future implementations of GridFTP servers. The plugin code will be available via the GridLab project software distribution, and will be published at <http://www.gridlab.org/>.

For benchmarking the software we used a dual Xeon 1.7GHz Server running RedHat Linux 8.0 as a data server. The machine was equipped with 1GB of RAM and a logical volume storage of 320 GByte (36.5 MByte/sec transfer rate). The measurements have a granularity of 1 second.

The visualization modules we described have been implemented in the Amira visualization environment [28, 5], which is based on OpenGL and OpenInventor. The renderings have been performed on a dual Pentium IV system with 2.6 GHz, 1 GByte main memory and NVidia Quadro4 graphics. The system ran under RedHat Linux 8.0 with the standard NVidia video driver.

7.2. Benchmark Results. In order to evaluate our approach, we performed a number of performance measurements for accessing, loading and displaying large remote HDF5 data sets. We compare the performance obtained using the GridFTP plugin (*GridFTP HDF5*) with a comparable remote access technique, that is HDF5 over GridFTP partial file access (*GridFTP PFA*). We also include measurements of local (*local access*) and Network File System (*NFS access*) times to see if we achieved our goal of having acceptable waiting times before the first visualization is created, considering the local and NFS times as acceptable.

The results of these tests are listed in table 7.2. The time needed to create the first image (t_3) is composed of the time needed to gather and transfer the meta data (t_1) and the time needed to filter and transfer the subsampled first timestep (t_2). t_4 gives the access time for a full resolution time step.

The tests have been performed on a Local Area Network (*LAN*) with normal network load (latency 1ms, measured 32.0 MBit/sec), and on a Wide Area Network connection (*WAN*) between Amsterdam and Berlin (latency 20ms, measured bandwidth: 24.0 MBit/sec).

The *WAN* measurements have been performed with various *level* settings, that is with different depth of the octree hierarchy created.

TABLE 7.1

The table lists performance measurements for the various access techniques we explored. The results have been obtained by timing the visualization process for a 32 GB HDF5 file, containing 500 timesteps, each timestep with the resolution of 256^3 data points (double precision).

Access Type	Net	Level	Meta Data t_1	Root Block t_2	Startup $t_3 = t_1 + t_2$	Complete t_4
local access	-	2	7 sec	1 sec	8 sec	3 sec
NFS access	LAN	2	8 sec	5 sec	13 sec	8 sec
GridFTP HDF5	LAN	2	11 sec	2 sec	13 sec	11 sec
GridFTP PFA	LAN	2	165 sec	10 sec	175 sec	200 sec
GridFTP HDF5	WAN	3	14 sec	2 sec	16 sec	126 sec
GridFTP HDF5	WAN	2	14 sec	3 sec	17 sec	68 sec
GridFTP HDF5	WAN	1	14 sec	7 sec	21 sec	45 sec
GridFTP HDF5	WAN	0	14 sec	41 sec	55 sec	41 sec
GridFTP PFA	WAN	3	430 sec	28 sec	458 sec	3760 sec
GridFTP PFA	WAN	2	430 sec	53 sec	483 sec	960 sec
GridFTP PFA	WAN	1	430 sec	110 sec	560 sec	477 sec
GridFTP PFA	WAN	0	430 sec	220 sec	670 sec	220 sec

These measurements show that the goal of a fast initial visual representation of the data set was achieved: a small startup time t_3 can be achieved by using the *GridFTP HDF5* technique combined with hierarchical access ($level \geq 2$). This time is of the same order of magnitude as for local visualization.

Specifying the hierarchy level provides the user with an interactive mechanism for tuning response times. The data access scheme could prove its adaptivity for different network connectivity. In principle, the user can reduce the time to obtain a first visual representation by choosing a larger hierarchy level. The tradeoff for

shorter startup times is the total transfer time for a fully resolved data set (all octree levels)⁷. The results show that relation (t_3 / t_4) clearly for the WAN measurements with different level settings.

Also, the large overhead for the complicated meta data access was dramatically reduced in comparison to GridFTP partial file access. The remaining time difference relative to the NFS meta data access results from the application of the zero filter to all data sets, the time needed to write the meta data file, and the time to transfer it.

8. Conclusions. With the presented scheme for progressive remote data access and its use for hierarchical rendering, we have successfully realized the functionality targeted in our motivating scenario (sect. 2). In particular, the techniques we have developed support the adaptation of remote data access to a wide range of I/O connections, and react flexibly to user and application demands. For example, our mechanisms support adjustment of the systems reaction time—the time until the first visual impression for the data set appears—by adapting data filter parameters, such as the chosen octree depth.

Our presented solution does not depend on server-side offline preprocessing of the complete data set. The access to the data sets meta data, when compared to naive remote access techniques, offers very high performance, as supported by the results of Table 1. Only a small local disk storage space is required for caching the associated metadata.

The extensibility of this approach is also notable. This approach supports both additional data formats other than HDF5, and access patterns other than hyperslab, through the provision of additional plugins. Simultaneously, it is important to acknowledge that this approach may make it increasingly difficult to maintain compatible configurations on all hosts of a Grid. The situation may improve with future GridFTP server implementations allowing dynamic linking and invocation of plugins. Thus implementation is one of the first few existing utilizations of the ERET capabilities provided by GridFTP. It is expected to see many more in the future.

Our work further demonstrates the usability of the data access scheme for hierarchical rendering techniques. The implemented algorithms (orthoslice, height field, volumetric rendering) show very good performance, and are also adaptive to user specification and connectivity characteristics.

The presented architecture enables us to realize visualization scenarios which would be impossible earlier, by reducing the total amount needed for obtaining a visual data impression by orders of magnitudes, if compared to naive approaches.

We are planning to enhance the dynamic protocol selection feature of Stork, so that it will not only select any available protocol to perform the transfer, but it will select the best one. The requirements of ‘being the best protocol’ may vary from user to user. Some users may be interested in better performance, and others in better security or better reliability. Even the definition of ‘better performance’ may vary from user to user. We are looking into the semantics of how to define ‘the best’ according to each user’s requirements.

We are also planning to add a feature to Stork to dynamically select which route to use in the transfers and then dynamically deploy DiskRouters at the nodes on that route. This will enable us to use the optimal routes in the transfers, as well as optimal use of the available bandwidth throughout that route.

Acknowledgments. It is a pleasure to thank many colleagues and collaborators who contributed to this work both directly and indirectly. At ZIB, that are namely Werner Bengler and Tino Weinkauff, who contributed to the overall ideas of our approach. We wish to thank the Globus group, in particular Bill Allcock and John Bresnahan, for their substantial support with the GridFTP server plugin infrastructure and implementation – without the experimental server provided by them, our work would have been hardly possible. We also wish to thank John Shalf and Werner Bengler for many insightful discussions about data handling. Finally, we wish to thank the members of the GridLab project who contributed to the Adaptive Component work package for useful discussions about (future) semi-automatic adaptivity schemes, and for their support during the benchmarking

The presented work was funded by the German Research Network (the DFN GriKSL project, grant TK-602-AN-200), and by the European Community (the EC GridLab project, grant IST-2001-32133).

REFERENCES

- [1] *HDF5 File Format Specification*, National Center for Supercomputing Applications (NCSA). <http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.format.html>.

⁷The maximum amount of additionally transferred data caused by the octree based access scheme is on the order of 15%. The higher number of resulting block requests increases the *overall* transfer time also due to the additional latencies.

- [2] *Introduction to HDF5*, National Center for Supercomputing Applications (NCSA). <http://hdf.ncsa.uiuc.edu/HDF5/doc/H5.intro.html>.
- [3] *Wide Area File Service and the AFS Experimental System*, Unix Review, 7 (1989).
- [4] *Network Programming Guide*, Sun Microsystems Inc., (1990). Revision A.
- [5] *Amira User's Guide and Reference Manual and Amira Programmer's Guide*, Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB) and Indeed-Visual Concepts GmbH, Berlin, 2001. <http://www.amiravis.com/>.
- [6] *Top 500*, (2002). <http://www.top500.org/list/2002/11/>.
- [7] *VisIt User's Manual*, Tech. Report UCRL-MA-152039, Lawrence Livermore National Laboratory, February 2003.
- [8] J. AHRENS, C. LAW, W. SCHROEDER, K. MARTIN, AND M. PAKKA, *A Parallel Approach for Efficient Visualizing Extremely Large, Time-Varying Datasets*, Tech. Report LAUR-00-1620, Los Alamos National Laboratory (LANL), 2000.
- [9] W. ALLCOCK, J. BESTER, J. BRESNAHAN, S. MEDER, P. PLASZCZAK, AND S. TUECKE, *Gridftp: Protocol extensions to ftp for the grid*, GWD-R (Recommendation), (2002). Revised: Apr 2003, <http://www.isd.fnal.gov/gridftp-wg/draft/GridFTPRev3.htm>.
- [10] G. ALLEN, W. BENGER, T. GOODALE, H.-C. HEGE, G. LANFERMANN, A. MERZKY, T. RADKE, E. SEIDEL, AND J. SHALF, *The Cactus Code: A Problem Solving Environment for the Grid*, in Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing, 2000, pp. 253–260.
- [11] D. ALPERT, *Scalable MicroSupercomputer*, Microprocessor Report, 03/17/03-01 (2003).
- [12] W. BENGER, I. FOSTER, J. NOVOTNY, E. SEIDEL, J. SHALF, W. SMITH, AND P. WALKER, *Numerical relativity in a distributed environment*, in Proceedings of the Ninth SIAM Conference on Parallel Processing for Scientific Computing, 1999.
- [13] W. BENGER, H.-C. HEGE, A. MERZKY, T. RADKE, AND E. SEIDEL, *Efficient Distributed File I/O for Visualization in Grid Environments*, Tech. Report SC-99-43, Zuse Institute Berlin, January 2000.
- [14] M. BEYNON, R. FERREIRA, T. KURC, AND J. SALTZ, *DataCutter: Middleware for Filtering Very Large Scientific Datasets on Archival Storage Systems*, in The Eighth Goddard Conference on Mass Storage Systems and Technologies/17th IEEE Symposium on Mass Storage Systems, College Park, Maryland, USA, March 2000.
- [15] B. CABRAL, N. CAM, AND J. FORAN, *Accelerated volume rendering and tomographic reconstruction using texture mapping hardware*, in 1994 Symposium on Volume Visualization, A. Kaufman and W. Krueger, eds., 1994, pp. 91–98.
- [16] T. CULLIP AND U. NEUMANN, *Accelerating volume reconstruction with 3D texture mapping hardware*, Tech. Report TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993.
- [17] I. T. FOSTER, C. KESSELMAN, G. TSUDIK, AND S. TUECKE, *A security architecture for computational grids*, in ACM Conference on Computer and Communications Security, 1998, pp. 83–92.
- [18] L. A. FREITAG AND R. M. LOY, *Adaptive, multiresolution visualization of large data sets using a distributed memory octree*, in Proceedings of SC99: High Performance Networking and Computing, Portland, OR, November 1999, ACM Press and IEEE Computer Society Press.
- [19] H.-C. HEGE AND A. MERZKY, *GriKSL—Immersive Überwachung und Steuerung von Simulationen auf entfernten Supercomputern*, DFN-Mitteilungen, 59 (2002), pp. 5–7.
- [20] F. ISAILA AND W. F. TICHY, *Mapping functions and data redistribution for parallel files*, in Proceedings of IPDPS 2002 Workshop on Parallel and Distributed Scientific and Engineering Computing with Applications, April 2002. Fort Lauderdale.
- [21] L. LINSEN, J. GRAY, V. PASCUCCI, M. A. DUCHAINEAU, B. HAMANN, AND K. I. JOY, *Hierarchical Large-scale Volume Representation with '3rd-root-of-2' Subdivision and Trivariate B-spline Wavelets*, Mathematics + Visualization, Springer Verlag, Heidelberg, Germany, 2003.
- [22] A. NORTON AND A. ROCKWOOD, *Enabling View-Dependent Progressive Volume Visualization on the Grid*, IEEE Computer Graphics—Graphics Applications for Grid Computing, (2003), pp. 22–31.
- [23] C. NUBER, R. W. BRUCKSCHEN, B. HAMANN, AND K. I. JOY, *Interactive visualization of very large datasets using an out-of-core point-based approach*, in Proceedings of the High Performance Computing Symposium 2003 (HPC 2003), I. Banicescu, ed., San Diego, California, March 30–April 2, 2003 2003, The Society for Computer Simulation International. Orlando, FL.
- [24] S. OLBRICH, T. WEINKAUF, A. MERZKY, H. KNIPP, H.-C. HEGE, AND H. PRALLE, *Lösungsansätze zur Visualisierung im High Performance Computing und Networking Kontext*, in Zukunft der Netze - Die Verletzbarkeit meistern., J. von Knop and W. Haverkamp, eds., vol. 10, Düsseldorf, Germany, May 2002, pp. 269–279. 16. DFN-Arbeitstagung über Kommunikationsnetze, GIEdition, Lecture Notes in Informatics (LNI).
- [25] N. RAMAKRISHNAN AND A. Y. GRAMA, *Data Mining Applications in Bioinformatics*, in Data Mining for Scientific and Engineering Applications, Kluwer Academic Publishers, 2001, pp. 125–140.
- [26] J. RUMBLE JR., *Publication and Use of Large Data Sets*, Second Joint ICSU Press - UNESCO Expert Conference on Electronic Publishing in Science, (2001). <http://users.ox.ac.uk/icsuinfo/rumblepr.htm>.
- [27] H. SCHUMANN AND W. MÜLLER, *Visualisierung*, Springer, Berlin, Heidelberg, New York, 2000.
- [28] D. STALLING, M. WESTERHOFF, AND H. HEGE, *Amira—an object oriented system for visual data analysis*, in Visualization Handbook, C. R. Johnson and C. D. Hansen, eds., Academic Press, to appear 2003. <http://www.amiravis.com/>.
- [29] M. WEILER, R. WESTERMANN, C. HANSEN, K. ZIMMERMAN, AND T. ERTL., *Level-of-detail volume rendering via 3D textures*, in IEEE Volume Visualization and Graphics Symposium 2000, 1994, pp. 7–13.
- [30] R. WOLSKI, N. SPRING, AND J. HAYES, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Journal of Future Generation Computing Systems, 15 (1999), pp. 757–768.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 5, 2003.

Accepted: September 1, 2003.



AN ADAPTIVE FILE DISTRIBUTION ALGORITHM FOR WIDE AREA NETWORK

TAKASHI HOSHINO* , KENJIRO TAURA* , AND TAKASHI CHIKAYAMA*

Abstract. This paper describes a data distribution algorithm suitable for copying large files to many nodes in multiple clusters in wide-area networks. It is a self-organizing algorithm that achieves pipeline transfers, fault tolerance, scalability, and an efficient route selection. It works in the presence of today's typical network restrictions such as firewalls and Network Address Translations, making it suitable in wide-area setting. Experimental results indicate our algorithm is able to automatically build a transfer route close to the optimal. Propagation of a 300MB file from one root node to over 150 nodes takes about 1.5 times as long as the best time obtained by the manually optimized transfer route.

Key words. Self-stabilizing distributed algorithm, fault tolerance, scalability, wide-area network

1. Introduction. This paper describes a practical algorithm for copying large data (typically in a file) from a source node(s) to many destination nodes in parallel. We seek a scalable solution suitable both within a cluster and across many clusters in wide-area. By suitable within a cluster, we mean that it fully utilizes the available bandwidth of LAN/cluster interconnect. For example, assuming 32 nodes are connected via a sufficiently high-throughput switch, it should be able to copy a single large file to the 32 nodes in not much more than the time it takes to copy the file to a single node. Such an algorithm must at least perform many one-to-one transfers in parallel. By suitable in wide-area, we mean it makes a good choice in selecting transfer routes. If many nodes in a cluster retrieve data from another cluster, a link across the two easily saturates. Thus such an algorithm should have a mechanism to transfer data within a cluster where possible.

To be practical, it should work with a simple and small manual configuration that may not be very accurate. It won't be practical to assume, for example, that the user gives a complete and accurate information about physical network topology, desirable paths for transferring data, or even logical network connectivity (i.e., network settings such as firewall and Network Address Translation (NAT)). Assuming such information is not practical not only because the user may not want to write them, but also because such information may change over time due to such events as node/network failures. The system therefore must tolerate inaccurate information and adapt to the conditions observed at runtime. Such an adaptive system naturally supports fault tolerance in the sense that even if some nodes fail, remaining nodes accomplish their work and nodes that once failed can join the transfer again.

We believe such a fault-tolerant and adaptive file replicator is a mandatory building block for cluster and Grid computing. It may be used for installing large program/data to many nodes. It may also be used in file synchronizers [5] so they support synchronizing data among a large number of nodes in parallel. Perhaps most important, replicating a large data to many nodes will be a practical technique to “reset” a distributed computation; it simply reinitializes all the involved nodes, so as to recover from some broken/inconsistent states. This observation accords with recent practices in large-scale cluster management, where reinstalling operating systems from scratch is considered as a normal operation, rather than the last resort, to fix broken clusters [13].

To get an intuitive idea about how a good transfer route typically looks, consider a network in Figure 1.1.

There are two local area networks (LANs) named A and B , each including three clusters ($A_1, A_2,$ and A_3 in A and $B_1, B_2,$ and B_3 in B). Assume nodes can connect to each other via the TCP layer.

Suppose the data is on a node in cluster A_1 and should propagate to all other nodes. In the figure, a small circle is a node, a rectangle a switch, and a line connecting a node and a switch a network cable that can transfer data with 100Mbps.¹

Intuitively, the best strategy is to form a transfer route like the one shown in Figure 1.2. Figure 1.3 represents the same route in the physical topology. Specifically, the following two properties are important.

- The number of connections that cross a LAN/cluster boundary is small; there is only one connection across the two LANs and five connections across the six clusters.
- The entire transfer route forms a list. That is, no nodes serve data to two or more nodes.

The reason why the first property is important will be clear. A simple calculation will reveal that if nodes are randomly connected without any effort to connect nodes close to each other, links across LANs/clusters will

*University of Tokyo, {hoshino,tau,chikayama}@logos.t.u-tokyo.ac.jp

¹Of course, this limit may not be due to the capacity of the cable *per se*, but due to NIC or switch.

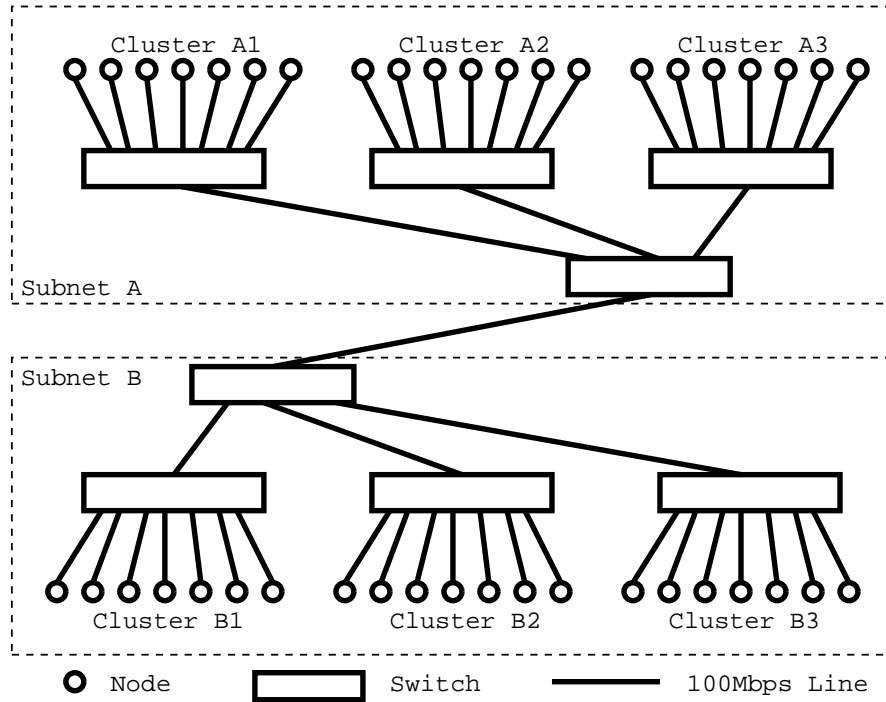


FIG. 1.1. Typical network environment for which our solution is suitable

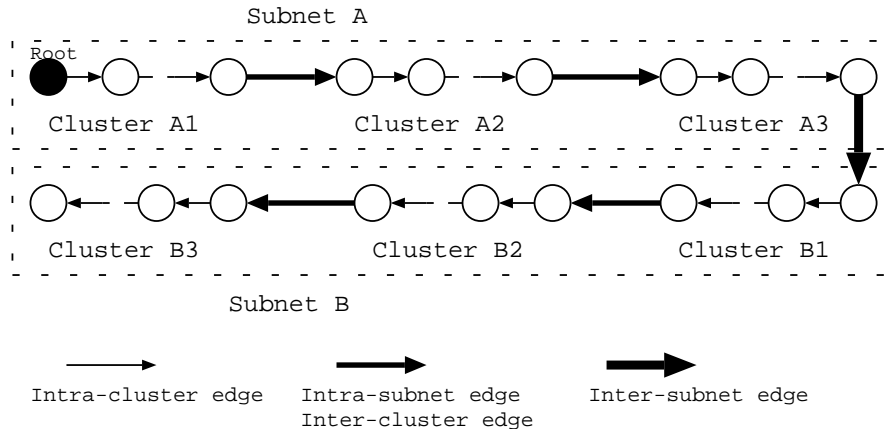


FIG. 1.2. Best transfer route in application layer

easily become a bottleneck. This is especially true in today’s typical network configuration where capacity of long links (corporate-/campus-/wide- area) is similar to or at best only an order of magnitude larger or so than typical local area links. For example, let us assume for the purpose of discussion that we have two 100Mbps switched LANs connected via a 1Gbps link. In such settings, we should be able to transfer data among all the nodes in the two LANs approximately at the LAN bandwidth (100Mbps), but if connections are randomly chosen, a link across the two LANs can sustain only 10 such connections at best. Thus the 1Gbps link won’t be enough for supporting 10 or more nodes in each side of it.

The second bullet may be less obvious. It is important for reducing the bottleneck in NICs. Suppose three nodes *A*, *B*, and *C* are linked via a 100Mbps switch. If data go from *A* to *B* to *C*, the throughput will be close to 100Mbps. If, on the other hand, *A* sends data both to *B* and *C* simultaneously, it can emit data at 50Mbps to each. Note that we assume *A* must send data to *B* and *C* separately, which we believe is a reasonable assumption because *B* and *C* may want different portion of the entire data stream. This is important especially

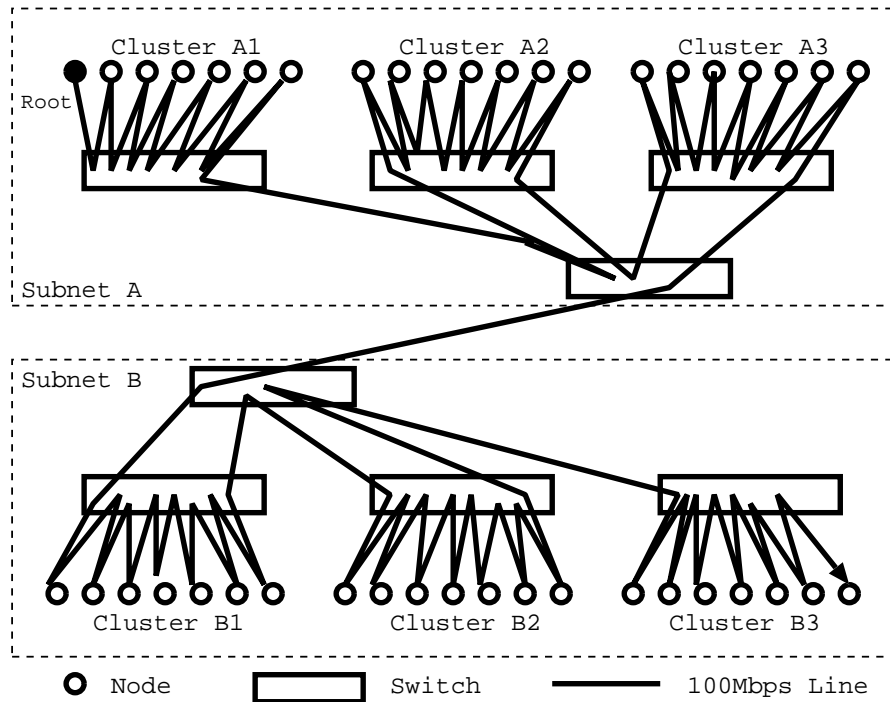


FIG. 1.3. *Best transfer route according to our guidelines in typical network*

when links across LANs are sufficiently powerful, so they won't become bottlenecks as long as we maintain the first property.

Our algorithm tries to build a transfer route close to such best routes. Note that it is not always possible to connect all nodes in a list. For example, if firewalls do not allow some connections, it may be unavoidable for some nodes to serve data to two or more children. Thus, our algorithm in general forms a transfer *forest*, with some heuristics to connect nodes close to each other and to make the tree deeper. It may be a forest, rather than a single tree, because there may be multiple nodes that have complete data in the beginning. In such cases, a separate tree will be formed rooted at each source node.

The paper is organized as follows. Section 2 describes a model of the network and the goal of this research. Then, we propose our algorithm and proof of efficiency in Section 3. And validation and evaluation are shown in Section 4. In Section 5, we explain related work. Finally, we conclude and summarize this research and remark to future work in Section 6.

2. Problem Description. In this section, we define goals of the algorithm and formalize the problem.

2.1. Goals.

Tolerate faults and adapt to resource conditions: Copying a large file to many nodes takes a long time. Therefore our solution must tolerate temporal/permanent network faults and node crashes. When a node crashes, nodes receiving data from the crashed node must find a substitute so that the remaining nodes finish their tasks. When a node recovers, it must be able to join the transfer network and continue its job, without waiting for the ongoing operation to finish and then restarting from scratch. In addition to being fault-tolerant, it must adapt to changes in network conditions; it should change the transfer route depending on changes of conditions.

Both of these requirements preclude a simplistic solution that statically constructs a route in the beginning and tries to retain the same route until they finish. Nodes must continuously search for a better transfer route.

Make an efficient transfer route automatically: As motivated in Section 1, our general criteria for "good" transfer route are (1) using a small number of "long" connections (i.e., connections that travel a large number of hops, such as inter-subnet connections), and (2) having a small number of nodes that serve

data to multiple (more than one) children. This is based on our assumption that a bottleneck is typically caused by an inter-subnet edge or a node. Examples for the latter are disks and network interfaces. Our algorithm tries to optimize the number of long connections and the number of children for each node, with a very simple local search heuristics.

Work on today's typical network configurations: Today's typical network configurations do not allow each node to connect to all other nodes. Firewalls may block connections between LANs. Inside a LAN, it is common to place all cluster nodes but one (a master node) behind a NAT router, so that accesses to clusters need go through the master. With DHCP, it may even be impractical to assume all nodes to have persistent names.

In short, we must model the network as a general graph where allowed connections are represented by its edges. Yet it is impractical to assume such a graph is given by the user (or the administrator) either offline or in the beginning of the algorithm. Altogether, we must design an algorithm that begins with a minimum amount of global information (e.g., participating nodes) and a local knowledge of the network (e.g., neighbors) in each node.

Do not assume physical network topology: Knowing physical network topology would help us to optimize transfer routes. Designing the algorithm assuming a complete knowledge about it is, however, impractical for many reasons discussed so far. First it is cumbersome for the user or the administrator to maintain such information. We may be able to obtain such information by using tools such as traceroute, but such tools tend to be unavailable these days for security considerations. It is also difficult to obtain the topology of the network behind a single router with traceroute. Second, even if topology information is available, dynamically probing the network is always necessary to make the algorithm fault-tolerant and adaptive. Algorithms based on probing connectivity and proximity at runtime naturally work without detailed knowledge about network topology.

Of course, we could always use physical topology as hints to our algorithm, among many other hints such as IP address prefix, latency, and observed throughput.

To achieve these goals, each node involved in our algorithm continuously seeks a parent, a node that serves data to the node. When it faces such events as parent crashes or disconnections, it tries to find a new parent. Even without such events, they continuously search for a better parent to optimize the transfer route. The criteria for a better parent are that (1) the closer a node is to itself the better, and (2) the fewer children a node has the better.

Our algorithm is a simple local search algorithm that converges to a satisfactory transfer in typical network configurations of today. Ideally, we desire an algorithm to find a globally optimal solution for any given network. A plausible definition of the optimal would be to minimize the sum of selected edge weights and the number of branches (or equivalently, the number of leaves) in the graph. The two criteria may conflict for general weighted graphs and even if they do not, they will require a complex global optimization algorithm (e.g., fault-tolerant MST construction) whose practical importance may not be very clear. In the following, we formulate our problem and prove our simple algorithm has a property which translates to "a sufficiently good" transfer route in typical real network configurations.

2.2. Problem Formulation. As usual, we model the network by a directed graph $G = \langle V, E \rangle$, where V is a set of nodes participating in the replication. E represents possible connections between nodes; $(a, b) \in E \iff a$ knows b 's name and the current network status allows a to connect to b .

The graph is for modeling purposes only; in practice, the network status may change over time, so each node cannot know the complete status of the network. It may even be impractical to assume each node knows all the neighbors it can connect to. In our implementation, each node begins with knowing information about a few of its neighbors and receiving a command that instructs it to participate in the replication of a file. They learn other node names on the fly by propagating information along established connections. This way, they learn other connections they may be able to make. They learn whether a particular connection is allowed or not by trying to establish a connection only when necessary. Nodes never maintain information about edges they are not adjacent to.

Below, we prove our optimization algorithm eventually reaches a transfer forest that has some desirable properties, assuming that the graph is fixed at some point. Note that our algorithm correctly finishes its job without this assumption. The assumption is essential only for stating the property of the forest our algorithm converges to.

To define the “goodness” of a transfer forest, we must introduce a notion of distance between nodes. One plausible formulation would be to give edges arbitrary weights, and to aim at reducing the total weights of selected edges (i.e., minimum spanning forest). We do not use this formulation but introduce a stronger assumption about the distances between nodes which we believe is a practical approximation of real networks, and show a simpler local search obtains sufficiently good results.

We assume nodes can be decomposed into groups so that nodes close to each other constitute a group. Our optimization algorithm does not assume that each node knows the decomposition explicitly, but only assumes that each node can somehow compare relative distances from the local node to other nodes. We show in Section 3.3 such a comparison induces a decomposition. It is such a decomposition for which our algorithm tries to reduce the number of inter-group edges. Again, the replication correctly finishes with inaccurate information, thus an implementation can use any sufficiently accurate measurement. Our current implementation is given in Section 3.2.1.

We say a decomposition is *complete* if nodes in each group form a clique (a complete subgraph) of G . That is, nodes inside a group can connect to each other without being blocked by, e.g., firewalls. For any decomposition which may or may not be complete, one can derive a complete decomposition by dividing its incomplete group into a number of groups so each of them is a clique. We call such a complete decomposition a *complete subdivision* of the original decomposition. Given a decomposition D , a complete subdivision that has the minimum number of groups is called the *coarsest* subdivision of D .

Given a decomposition, the goal would be to make a transfer forest close to the following *best desirable*, which has

1. the minimum number of edges connecting nodes in different groups, and
2. the minimum number of branches.

Our algorithm converges to the optimal if each node can connect to any other node (i.e., the entire graph is complete, or in practical terms, firewall, NAT, or DHCP do not deny any connection against us). In more general graphs, our algorithm has the following property. Let D the decomposition induced by a heuristics used to measure the relative distance between nodes, and \overline{D} the coarsest complete subdivision of D . Our algorithm achieves (1) the number of inter-group connections $\leq \overline{N} - F$ and (2) the number of branches $\leq \overline{N} - 1$, where \overline{N} is the total number of groups in \overline{D} and F the number of groups in \overline{D} containing at least one *finished* node, a node which has received the entire data.

Our claim that the above property translates to a good result in practice is based on the following observations.

- A simple measurement can reasonably approximate the “closeness” between nodes. For example, given a node in the same LAN as the local node and another not in the same LAN, it will be relatively easy for the local node to judge if one node is closer to the other, thus should be preferred. Therefore, one can obtain a decomposition each group of which has nodes close to each other.
- In typical network configurations, nodes close to each other tend to be allowed to connect to each other. Most typically, nodes within a LAN can connect to each other. Making a group of nodes close to each other thus tends to yield a subgraph that is nearly complete.

The first bullet implies that, if we group nodes based on a reasonably accurate measurement of distances between them, we will have groups each of which consists of nodes close to each other. Each such group will be nearly complete (bullet #2), therefore \overline{N} will be close to N . Together, the number of connections crossing a group boundary will be close to $N - F$, and the number of branches close to $N - 1$.

3. Algorithm. The algorithm has several features that we should remark.

A simple, self-stabilizing distributed algorithm: Each node works based on information about its neighbors and optimizes transfer routes with a small amount of local information. Each node *continuously* seeks a closer node that may serve data faster. This mechanism naturally makes our algorithm fault-tolerant and allows nodes to join or leave computation at any time.

Parallel and pipelined transfer: Transferring data from node A to B and from C to D can occur in parallel. Moreover, transferring a piece of data from A to B and transferring another piece of data from B to C can also take place in parallel (pipelined transfer). This is especially important for replicating large files in switched networks.

A simple transfer loop avoidance: The algorithm naturally avoids deadlock due to a transfer loop simply by letting each node become a parent of another only when it has more data than others. This mechanism,

together with the self-stabilizing nature of the algorithm, is enough to make it deadlock-free; when a node crashes, its children will eventually learn there is no progress for a long time, in which case they try to connect to another node that is ahead of it.

```

01: /* Starting or After Recovered */
02: offset = current filesize on disk;
03: parent = invalid; /* the node self is getting data from.
   */
04: candidate = null;
05: is_sending_giveme = false;
06: children = none; /* nodes self is giving data to */
07: siblings = none; /* used for Tree2List Suggestion */
08: neighbors = list of neighbors (dead or alive);
09: while (true) {
10:   /****** Searching for Parent *****/
11:   (candidate == null && parent == invalid) =>
12:     candidate = a node in neighbors;
13:     send(candidate, ask(id, offset));
14:   /* NearParent Heuristics */
15:   (candidate == null && a node in neighbors satisfies
16:   is_closer(self, node, parent)) =>
17:     candidate = node;
18:     send(candidate, ask(id, offset));
19:   /* Tree2List Heuristics */
20:   (candidate == null && a sibling in siblings satisfies
21:   !is_closer(self, parent, sibling)) =>
22:     candidate = sibling;
23:     send(candidate, ask(id, offset));
24:   received(ask(wid, woffset)) =>
25:     if ((offset > woffset) &&
26:     (MAX_NODE > number of children)) {
27:       add this node (wid, woffset) to children;
28:       send(wid, ok(id, offset));
29:     } else {
30:       send(wid, ng(id));
31:     }
32:   received(ok(wid, woffset)) =>
33:     if (woffset > offset) {
34:       parent = wid; candidate = null;
35:     }
36:   received(ng(wid)) =>
37:     if (wid == candidate) {
38:       candidate = null;
39:     } else if (wid == parent) {
40:       parent = invalid;
41:     }
42:   /****** Data Transfer *****/
43:   (parent != invalid && offset < filesize &&
44:   !is_sending_giveme) =>
45:     is_sending_giveme = true;
46:     send(parent, giveme(id, offset));
47:   received(giveme(child, woffset)) =>
48:     if (offset > woffset) {
49:       size = max(BLOCKSIZE, offset - woffset);
50:       buf = load(filename, woffset, size);
51:       send(child, data(id, woffset, size, buf));
52:     } else {
53:       send(child, ng(id));
54:     }
55:   received(data(wid, woffset, size, buf)) =>
56:     if (woffset == offset) {
57:       is_sending_giveme = false;
58:       save(filename, woffset, size, buf);
59:       offset += woffset;
60:     }
61:   (offset == filesize && parent != null) =>
62:     if (parent != invalid)
63:       send(parent, disconnect(id));
64:     parent = null;
65:   received(disconnect(child)) =>
66:     delete the child from children;
67:   /****** Tree2List Suggestion *****/
68:   (having more than one child) =>
69:     foreach child in children {
70:       send(child, suggestion(id, children));
71:     }
72:   received(suggestion(parent, new_siblings)) =>
73:     siblings = new_siblings;
74:   /****** Fault Handling *****/
75:   (timeout(data, ng) from parent) =>
76:     parent = invalid;
77:   (timeout(giveme, disconnect) from child) =>
78:     delete the child from children;
79:   (timeout(ok, ng) from candidate) =>
80:     candidate = null;
81: }

```

FIG. 3.1. Pseudo-code of our algorithm

Figure 3.1 shows the local algorithm running on each node. Prior to running this algorithm, each node knows its neighbors (*neighbors*) and the size of the file each node must eventually have (*filesize*). In actual implementation, each node may begin with an incomplete list of neighbors. Nodes propagate their neighbors to other and learn from others.

Inside the main **while** loop (line 9–81) is written as a list of the following form:

condition => *action*

where *condition* is a precondition (or a *guard*) in which the *action* can take place. The predicate **received**(*X*) evaluates to true if a message that matches *X* is in the incoming message queue of the node. Each iteration of the loop waits for at least one guard to become true, and executes the corresponding action. If multiple guards

are true, any one of them is chosen arbitrarily.

First, we explain the base part of this algorithm in Section 3.1. We continue with the route optimization heuristics in Section 3.2

3.1. The Base Algorithm. Each node repeats the following until it gets the entire data.

- It seeks a node that is ahead of itself (i.e., has more data than itself). Let us call such a node *its parent*. A parent may change over time.
- Once it finds a parent, it asks the parent to send the data that should come next to the data it currently has. For example, if a node has the first 1000 bytes of a file, it will ask the parent to send some amount of data from offset 1000.
- In addition,
 - Each node, except ones that have obtained the entire data, seeks a node that is closer to its current parent. Details are in Section 3.2.1.
 - Each node having two or more children tries to resolve this situation, by suggesting children to connect to one of its siblings.

When a node receives an instruction to participate in a replication, each node checks how much data it has (line 2), searches for a candidate node that has data greater than itself by connecting to some nodes in its *neighbors* list. Variable *offset* indicates the size of data at that time, and satisfies the inequality $0 \leq \text{offset} \leq \text{filesize}$. During data transfer, the invariant *child's offset* \leq *parent's offset* is maintained (line 25, 33, and 48).

A node searching for a parent sends an *ask* message carrying its *offset* (data size) to a candidate (line 11–13). If the receiver has more data than the sender, it sends an *ok* message to the node sender (line 24–28, 32–35). At that time, the relation between parent-child is established. After that, the child sends a *give me* message to the parent (line 43–46) and the parent sends a chunk of data to the child (line 47–51). This repeats until the child either catches up the parent in data size (line 52–54), finds a better candidate than the current parent, or receives an error. If the receiver of *ask* does not have more data than the sender, it sends an *answer ng* (line 29–31) to the sender. Receiving an *ng* message (line 36–41), the node continues to search for a parent.

A node can be a parent of some nodes and a child of another at the same time. In effect, we achieve a pipeline transfer through all nodes.

When a parent becomes unreachable from its child (due to a parent crash or a network failure), the child merely searches for a new parent. When a node recovers, it can participate in the transfer from the offset at the time it has failed. Hence, this algorithm is fault-tolerant (line 74–80).

3.2. Adaptive Transfer Route Optimization. Now, we explain optimizing heuristics on top of the base algorithm (line 14–23, 67–73).

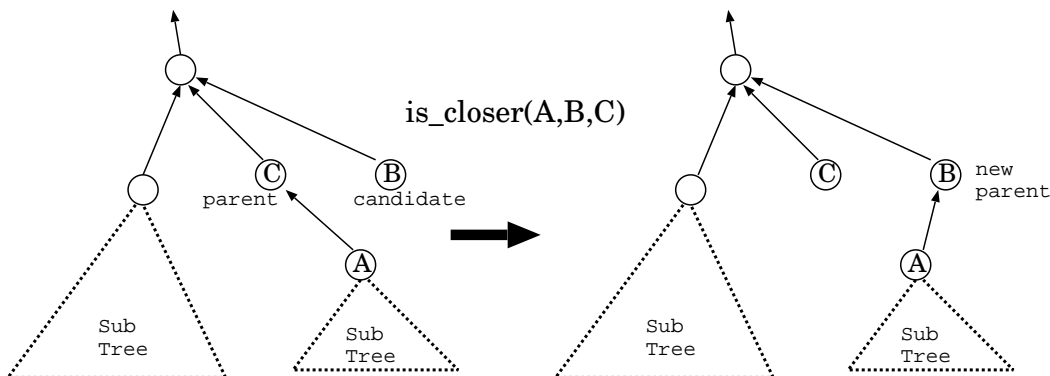


FIG. 3.2. NearParent Operation

3.2.1. NearParent Heuristics. Each node periodically tries to connect to a node that is closer to its current parent (*candidate* in Figure 3.2, line 15–18 in Figure 3.1). If the candidate node turns out to have more data than the local node (line 32–35), it selects the new node as the new parent. Figure 3.2 shows how this heuristics modifies a part of the transfer tree.

Note that even if each node has connected to its parent, it searches for an even closer candidate periodically. We have not conducted an extensive study about the best frequency. Frequent measurements will allow us to find a good transfer route fast at the cost of increased network traffic. Our current implementation guarantees that there is at most one traffic from each node for the measurement. It also guarantees each node performs a measurement at most once every 100ms. This will hardly affects CPU or network load.

The predicate to judge if a node B is closer than C from the local node A , $\text{is_closer}(A, B, C)$, currently uses the following criteria in the listed order.

Throughput observed in the past: Each node records throughput from each of the nodes that have been chosen as its parent. If A has chosen both B and C as its parent before, whichever produced a better throughput is considered closer.

Observed latency: The above criterion is not applicable when either B or C has never been chosen one as A 's parent. In this case A uses latencies it takes to connect to B and C .

The length of the matching IP address prefix: When observed latencies are too close to discriminate, we use IP addresses of A , B , and C . We compare the lengths of the common prefixes of IP addresses of A and B to that of A and C .

For the purpose of proving the theoretical property of the algorithm mentioned in Section 3.3 (also stated as Theorem 3.7), is_closer can be any predicate that satisfies the following properties.

- $\text{is_closer}(A, B, C)$ and $\text{is_closer}(A, C, B)$ do not become true at the same time.
- For a given A , the binary relation:

$$R_A(B, C) \stackrel{\text{def}}{=} \text{is_closer}(A, B, C)$$

is transitive. That is,

$$\begin{aligned} &\text{is_closer}(A, B, C) \wedge \text{is_closer}(A, C, D) \\ &\Rightarrow \text{is_closer}(A, B, D) \end{aligned}$$

- $\text{is_closer}(A, B, C) \Rightarrow \text{is_closer}(B, A, C)$

It will be clear that any reasonable definition of relative distance and an accurate measurement of it, including the ones listed above, will satisfy the first two bullets. The third property may not sound very obvious. Examples that satisfy the property include:

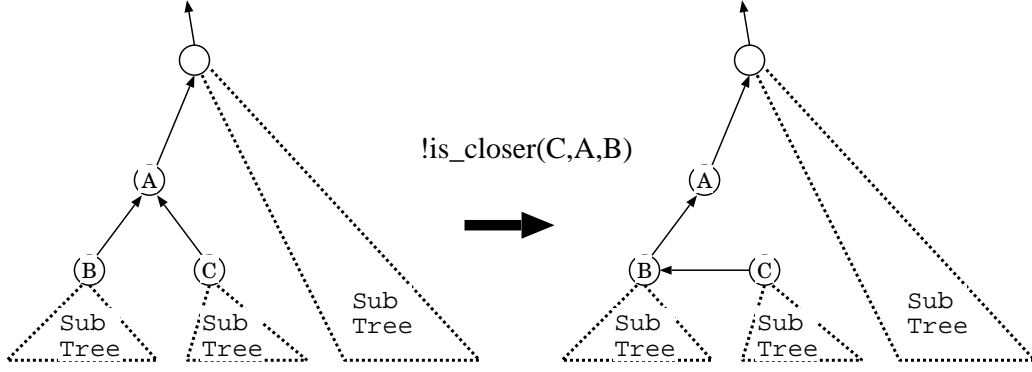
- A definition based on the bottleneck edge on trees. That is, assume nodes are connected via a weighted tree and let $\text{is_closer}(A, B, C)$ be true iff the minimum weight on the path between A and B is larger than that on the path between A and C .
- A definition based on the distance on trees. That is, assume nodes are connected via a tree and let $\text{is_closer}(A, B, C)$ be true iff the path between A and B is shorter than A and C .
- A definition based on address prefixes. That is, assume nodes are assigned integer addresses and let $\text{is_closer}(A, B, C)$ be true iff the length of the matching address prefix between A and B is larger than that between A and C .

Therefore we expect that our current implementation of is_closer based on measured bandwidths between nodes, measured latencies between nodes, and the length of IP address prefixes, will satisfy the third property provided measurements are accurate.

Note that implementing such a predicate does not require any *a priori* notion of groups. Just defining/measuring the relative closeness between nodes will suffice, as long as such a definition/measurement satisfies the above properties. In Section 2.2, we show such a predicate in general implicitly induces a distance between nodes, which in turn induces a decomposition of nodes based on the distance. Our algorithm reduces the number of inter-group edges for a decomposition derived this way.

3.2.2. Tree2List Heuristics. NearParent heuristics reduces the number of edges that cross group boundaries. It however is not useful for reducing the number of branches. Another optimization, called Tree2List heuristics, comes into play to make the transfer route closer to a list.

A node that has two or more children sends its children list to every child (line 68–71). When a node receives a suggestion message, which effectively contains its current siblings, it chooses one in the list as the next candidate if the current parent is not closer to it (lines 72–73, 20–23). Figure 3.3 shows how Tree2List heuristics modifies a part of the transfer tree. Intuitively, Tree2List pushes branches in a transfer tree downwards, hoping the tree eventually becomes a list.

FIG. 3.3. *Tree2List Operation*

An important property about Tree2List, proved in the next section, is that it never increases the number of inter-group edges. This guarantees that applying Tree2List does not impede the NearParent's effort of reducing the number of inter-group edges. In the next section, we state and prove properties of transfer forests after applying both heuristics in an arbitrary order.

3.3. Properties of the Route Optimization Algorithm. Let `is_closer` satisfy the properties stated in Section 3.2.1. We first show the following, that says `is_closer(A, B, C)` is equivalent to comparing a distance between A and B and between A and C , for some definition of a distance.

LEMMA 3.1. *For `is_closer` satisfying the property stated in Section 3.2.1, there exists a distance function d that satisfies the following.*

- For all nodes A and B , $d(A, B) = d(B, A)$.
- For all nodes A , B , and C ,

$$\text{is_closer}(A, B, C) \iff d(A, B) < d(A, C).$$

Proof: See Appendix A.1.

The following Lemma is important for guaranteeing Tree2List is applicable when we have many branches.

LEMMA 3.2. *For any d satisfying the condition in Lemma 3.1,*

$$\max(d(A, B), d(A, C)) \geq d(B, C)$$

is true for all nodes A, B , and C . Proof: See Appendix A.2.

A distance function d and a threshold t define a natural decomposition of a graph. That is, we remove all edges (x, y) such that $d(x, y) > t$ from the original graph, and let a group be a connected component of the graph. We call such a decomposition is *derived* from `is_closer`. Many decompositions can be derived from a single definition of `is_closer`, depending on the choice of d and t .

We model our route optimization heuristics as a process of rewriting the transfer forest according to NearParent, Tree2List, or finishing the transfer to a node.

DEFINITION 3.3. *A state of computation is a forest among participating nodes, induced by their parent pointers. Let S and S' be states. We define relations \rightarrow_n , \rightarrow_t , \rightarrow_f , and \rightarrow by:*

1. $S \rightarrow_n S' \stackrel{\text{def}}{\iff} S'$ is obtained by applying NearParent to S (Figure 3.2),
2. $S \rightarrow_t S' \stackrel{\text{def}}{\iff} S'$ is obtained by applying Tree2List to S (Figure 3.3),
3. $S \rightarrow_f S' \stackrel{\text{def}}{\iff} S'$ is obtained by finishing a node and making its parent pointer null, and
4. $\rightarrow \stackrel{\text{def}}{=} \rightarrow_n \cup \rightarrow_t \cup \rightarrow_f$. That is,

$$S \rightarrow S' \stackrel{\text{def}}{\iff} (S \rightarrow_n S') \text{ or } (S \rightarrow_t S') \text{ or } (S \rightarrow_f S').$$

Next, we define some quantities of states. Below, we fix a decomposition D derived by `is_closer`, and let \overline{D} be the coarsest subdivision of D . Let d and t the distance function and the threshold that induced D . Let \overline{N} be the number of groups in \overline{D} . When we say a group, it always means a group of \overline{D} . Nodes in a single group by definition form a clique.

DEFINITION 3.4.

- Let $w(S)$ be the number of edges in forest S that cross group boundaries. For technical convenience, we consider an invalid `parent` pointer to cross a group boundary, and a null `parent` pointer not to cross any group boundary.
- Let $f(S)$ be the number of finished nodes (having `parent` = null) and $F(S)$ be the number of groups that have at least one finished node. We say such a group is finished. Note there may be unfinished nodes in a finished group.
- Let $l(S)$ be the number of leaves (i.e., nodes that are not pointed to by any `parent` pointer).

LEMMA 3.5. *Transition paths are bounded. That is, the length of a path $S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \dots$ is bounded.*

Proof: Define $SUMDIST(S)$, $SUMDEPTH(S)$, and $Q(S)$ as follows.

$$\begin{aligned} SUMDIST(S) &= \sum_{x: \text{node}} d(x, x\text{'s parent}), \\ SUMDEPTH(S) &= \sum_{x: \text{node}} \text{depth}(x), \text{ and} \\ Q(S) &= (f(S), -SUMDIST(S), SUMDEPTH(S)), \end{aligned}$$

where $\text{depth}(x)$ is the number of hops from the root of the tree x belongs to. $d(x, x\text{'s parent})$ is the distance between x and its parent. Again for technical convenience, if $x\text{'s parent}$ pointer is invalid we consider it has a value larger than any other $d(y, z)$ for $z \neq \text{invalid}$. Similarly, if $x\text{'s parent}$ is null, it takes a value smaller than any other $d(y, z)$ for $z \neq \text{null}$.

If we introduce a lexicographical order among triples $Q(S)$, it is easy to see $Q(S)$ strictly increases by a single transition step. That is,

$$S \rightarrow S' \Rightarrow Q(S) < Q(S').$$

In fact, \rightarrow_f increases $f(S)$, \rightarrow_n does not change $f(S)$ and increases $-SUMDIST(S)$, and \rightarrow_t does not change $f(S)$, never decreases $-SUMDIST(S)$, and increases $SUMDEPTH(S)$.

Since all quantities of the triples are clearly bounded above, we have proved transition paths are bounded. \square

LEMMA 3.6.

1. If S satisfies $w(S) > \overline{N} - F(S)$, then \rightarrow_n is applicable to S . That is, there exists S' such that $S \rightarrow_n S'$.
2. If S satisfies $l(S) - f(S) \geq \overline{N}$, $f(S) \geq 1$, and \rightarrow_n is not applicable to S , then \rightarrow_t is applicable to S .

Proof:

1. If $w(S) > \overline{N} - F(S)$ (= the number of unfinished groups), either of the following must hold.
 - There is an unfinished group having more than one outgoing inter-group edges.
 - There is a finished group having an outgoing inter-group inter-group edge.

An *outgoing edge* is a `parent` pointer pointing to a node outside the group. In the former case, let two of such edges be (A, B) and (C, D) . A and C belong to one group, say X , while neither B nor D belong to X . Thus, a transition by \rightarrow_n that either makes A one of $C\text{'s}$ children or vice versa, is applicable. In the latter case, let one such edge be (A, B) and one finished node in the group be P . Thus, a transition by \rightarrow_n that makes A one of $P\text{'s}$ children is applicable.

2. We split the proof into two cases, (i) $l(S) - f(S) > \overline{N}$, and (ii) $l(S) - f(S) = \overline{N}$.

(i) $l(S) - f(S) > \overline{N}$:

We have at least one group X that satisfies:

$$l - f > 1$$

where l and f denote the number of leaves in X and the number of finished nodes in X , respectively. Let a_1, a_2, \dots, a_l be the leaves in X ($l \geq 2$). Let $a_{i,1} = a_i$ and $\vec{a}_i = (a_{i,1}, a_{i,2}, \dots, a_{i,n_i})$ ($i = 1, \dots, l$) be chains of `parent` pointers starting from a_i . That is, $a_{i,j}$ is a child of $a_{i,j+1}$ for all i and j ($1 \leq i \leq l$, $1 \leq j \leq n_i - 1$).

We argue by contradiction that all but one of such chains must be entirely in X . Let us assume w.o.l.g. neither of \vec{a}_1 nor \vec{a}_2 are in X . Then there are j and k ($1 \leq j \leq n_1 - 1$ and $1 \leq k \leq n_2 - 1$) such that $a_{1,j}$ and $a_{2,k} \in X$, and $a_{1,j+1}$ and $a_{2,k+1} \notin X$. Then a transition by \rightarrow_n that connects $a_{1,j}$ and $a_{2,k}$ should be applicable. This contradicts the assumption that \rightarrow_n is not applicable in S .

Now we have $l - 1$ chains entirely in X . Since $l - f \geq 2$ ($\Rightarrow l - 1 \geq f + 1$), at least two of them must merge at some node in X . Let a node at which two merges be A , and B and C the children of A on the

two chains. It remains to show we have either $(\neg \text{is_closer}(B, A, C))$ or $(\neg \text{is_closer}(C, A, B))$, so either B or C can trigger \rightarrow_t . By Lemma 3.2, we have

$$\max(d(A, B), d(A, C)) \geq d(B, C),$$

from which we can derive:

$$\begin{aligned} & \max(d(A, B), d(A, C)) \geq d(B, C) \\ \Leftrightarrow & d(A, B) \geq d(B, C) \text{ or } d(A, C) \geq d(B, C) \\ \Leftrightarrow & d(B, A) \geq d(B, C) \text{ or } d(C, A) \geq d(C, B) \\ \Rightarrow & \neg \text{is_closer}(B, A, C) \text{ or } \neg \text{is_closer}(C, A, B). \end{aligned}$$

(ii) $l(S) - f(S) = \overline{N}$:

If we have one group X that satisfies:

$$l - f > 1,$$

then the same discussion as (i) applies. In the remaining case all the groups satisfy:

$$l - f = 1.$$

Let X be any group. As in (i), consider the l chains starting from a node in X . If all the l chains are entirely in X , two of them must merge in X , and the following argument is the same as (i). Therefore each group has exactly one chain outgoing from the group. Then we have \overline{N} inter-group edges, i.e., $w(S) \geq \overline{N}$. This implies, however, \rightarrow_n is applicable because $f(S) \geq 1 \Rightarrow F(S) \geq 1 \Rightarrow w(S) \geq \overline{N} > \overline{N} - F(S)$.

□

THEOREM 3.7. *Along any path of state transitions starting from any state I , we reach within finite steps a state S_∞ satisfying:*

1. $w(S_\infty) \leq \overline{N} - F(S_\infty)$, and
2. $l(S_\infty) - f(S_\infty) \leq \overline{N} - 1$.

Proof: From Lemma 3.5, any transition path $I = S_0 \rightarrow S_1 \rightarrow \dots$ is bounded, therefore reaches a state S_∞ in which neither \rightarrow_n nor \rightarrow_t (or \rightarrow , for that matter) is applicable. Lemma 3.6 shows in this state we have both of the above properties. □

Remark 1: As a special case where $D = \overline{D}$ (i.e., no edges are blocked inside a group of D), we have $N = \overline{N}$. In this case the theorem implies that, for sufficiently long transfers, the number of edges between groups reaches the optimal $N - F(S)$. Replicating a file from $F(S)$ groups to the rest will clearly need $N - F(S)$ inter-group edges. For being close to a list, the second bullet of the theorem implies that the number of branches, effectively calculated by $l(S) - f(S)$, is the optimal $N - 1$. To see this is optimal in general, consider a network configuration shown in Figure 3.4, which forces inter-group edges to form a star.

Remark 2: Recall that the theorem applies to *any* decomposition derived from is_closer . If the network has multiple levels of hierarchies, (e.g., inside a cluster, clusters inside a LAN, LANs in a campus/corporate area, and LANs in wide area), and is_closer can discriminate all of them, our algorithm simultaneously optimizes all the levels. For example, let us say we have N_1 LANs and N_2 clusters and $f(S) = 1$ as the usual case. If we assume is_closer can discriminate intra-cluster, inter-cluster but intra-LAN, and inter-LAN edges, and the network configuration allows all connections, our algorithm converges to a state in which we have $N_1 - 1$ inter-LAN edges and $N_2 - 1$ inter-cluster edges.

4. Evaluation.

4.1. Implementation. We have implemented the described algorithm in Java. This is executable on common computers supporting Java and TCP/IPv4 protocol. We confirmed the program runs on Solaris (sparc), Linux (x86), Windows (x86), and Tru64Unix (Alpha). Stopping some nodes in the middle of a distribution task did not prevent any of the remaining nodes from finishing the task, confirming its fault-tolerance.

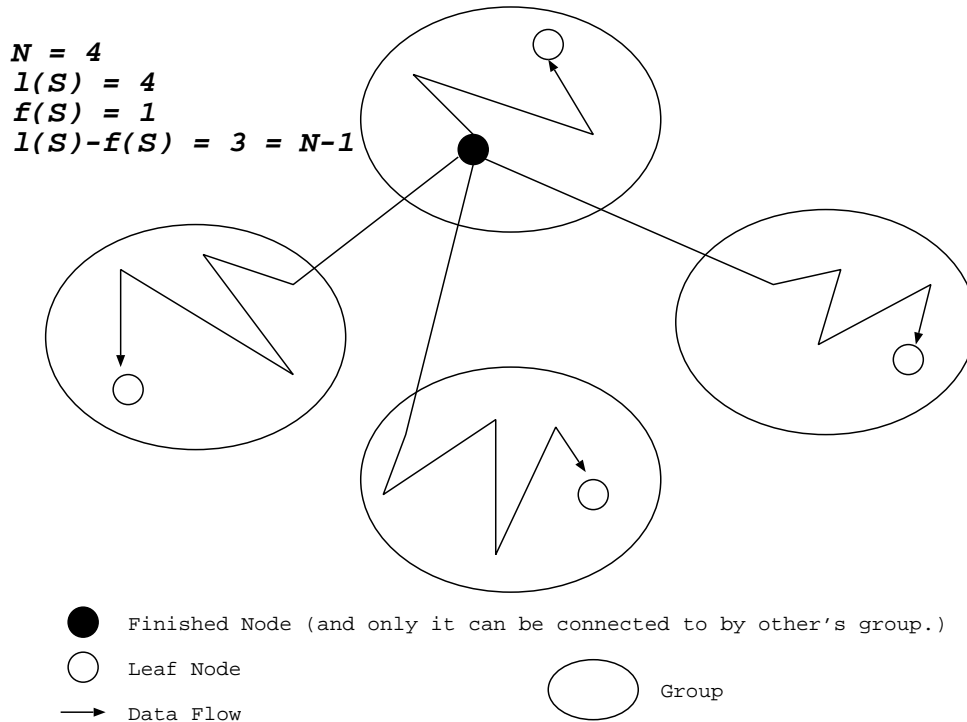


FIG. 3.4. An example where the optimal value of $l(S) - f(S)$ is $N - 1$

4.2. Single Cluster Experiments. First, we ran some experiments in a single cluster. The cluster consists of 16 nodes. Each node has two Alpha CPUs and a local hard-disk. Network cables of nodes are connected to a 100Mbps switch. Local disk bandwidth is faster than network, so it does not reveal as a bottleneck. CPU is also fast enough.

We initially let one node have 500MB file, and others have no data. Since there is only a single cluster, NearParent optimization does not play any role in this experiment. So this experiment is to see the effect of Tree2List. In addition to Tree2List, we ran the base algorithm without any optimization, changing the maximum number of children each node can serve, from one to five. They clearly demonstrate how important is it to make the transfer tree close to a list.

The time which the distribution tasks spent is shown in Figure 4.1.

In this result, it is clear that the average distribution time increases as the maximum number of children increases. The graph also indicates that, in this particular experiment, limiting the number of children to one yields the best result. That is, restricting the shape of the transfer tree to a list in the first place is better than our Tree2List strategy which first forms an arbitrary tree and then tries to develop it to a list. We believe, however, our strategy has several advantages. First, nodes may not be able to form a list in the presence of firewalls etc. In such cases, one must fall back to a tree. Second, forming a list in the beginning may take much longer than forming a tree, especially when the number of nodes becomes large, since a list can only grow one node at a time.

4.3. Multiple Cluster Experiments. Next, we made experiments in seven clusters illustrated in Figure 4.2. They are all placed in the campus of University of Tokyo.

- An IBM Linux cluster called “istbs” contains 70 nodes. We used all of them for the experiment. Nodes within a cluster are connected via 1Gbps links. A node in this cluster is the source node in this experiment. Bandwidth from/to other clusters below is poor 100Mbps.
- A SunFire15K SMP called “istsun” has 70 CPUs, of which we used 20. We used this machine as if it were 20 separate nodes. It has a 100Mbps NIC shared by all CPUs. Replication of 300MB data among 20 nodes inside istsun takes about 70 sec, where the throughput is about 34Mbps. This seems due to disk I/O bandwidth.

Plot of Experiments changing Children Limit in One Cluster

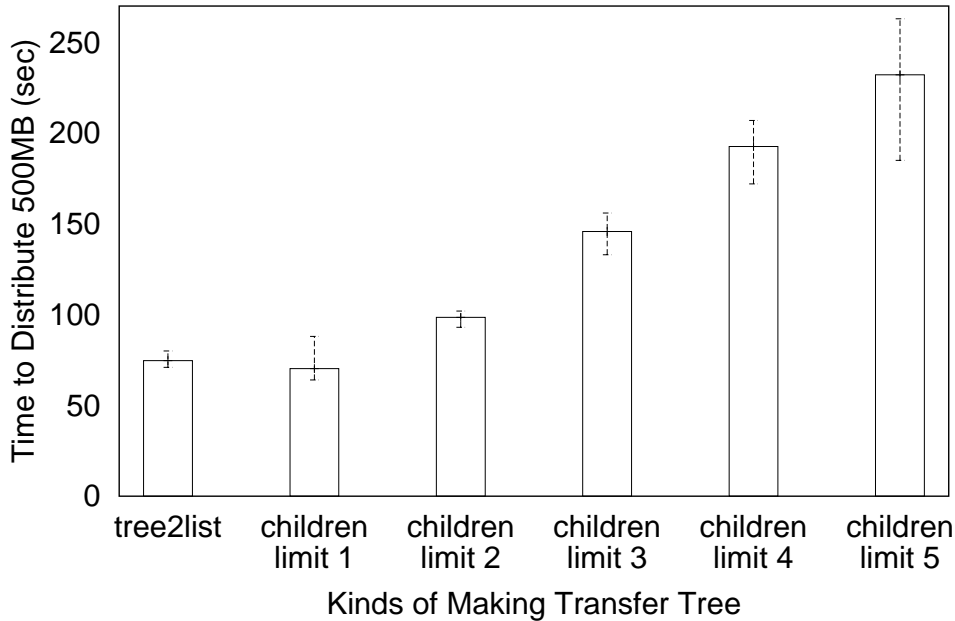


FIG. 4.1. Performance in a single cluster

- A cluster of clusters called “kototoi” contains three cluster each having 16 nodes. Network speed is 100Mbps inside each cluster. Throughput between two of the three is several hundreds Mbps. Having more than one connection to a single cluster easily saturates the link. No nodes outside kototoi cannot directly connect to inside it.
- An HP Alpha cluster called “oxen” contains 16 nodes, which is the same cluster in Section 4.2. There are two (and only two) gateway nodes that can connect to and can be connected from outside the cluster.
- A Linux cluster called “marten” each of which runs Linux inside VMWare. Its configuration is almost the same as a cluster in kototoi.
- For connectivity, any node can connect to istsun nodes and the gateways of oxen. Also, istsun and istbs are in the same virtual LAN, so nodes in the two clusters can directly connect to each other. Connections to remaining nodes from other clusters are blocked.

We compared the following algorithms.

Random tree: The base algorithm without any heuristics, with no limit on the number of children for each node.

NearParent only: The base algorithm + NearParent. No Tree2List.

Tree2List only: The base algorithm + Tree2List. No NearParent.

NearParent + Tree2List: Use both Tree2List and NearParent.

Manual: Fix the transfer route that we consider will be the best, as follows; istbs connects to istsun via one inter-cluster edge. It is branched into three inside istsun. They go to kototoi, oxen, and marten. Inside clusters, there are no branches. The throughput should be close to $100\text{Mbps} / 3 = 33\text{Mbps}$, determined by the three outgoing edges from istsun, which share a single 100Mbps NIC.

In Figure 4.3, the results are presented. Not surprisingly, “Manual” is the fastest. NearParent + Tree2List achieved an overhead of 50-100% to the manually tuned transfer and more than four times faster than the random tree.

Figure 4.4 shows that the number of inter-cluster edges and distribution time have a strong correlation.

This result confirms that reducing inter-cluster (and inter-subnet) edges strongly affects performance of replication among many nodes.

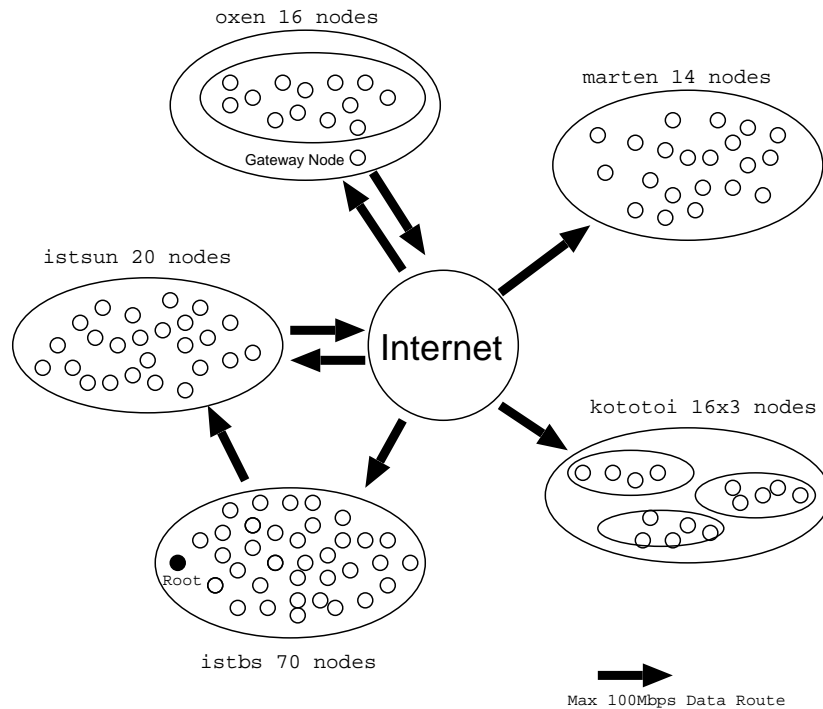


FIG. 4.2. Condition of 7 clusters

5. Related Work.

5.1. Minimum Spanning Tree Construction. MST construction is a commonly used technique for optimizing flows in networks. There have been a number of published algorithms and their applications [2, 6, 1]. It is compelling to model our problem by a general weighted graph, with the goal being a tree that has a small weight and a small number of branches.

We considered approaches along this line and then abandoned them for several reasons. First, from theoretical point of view, minimizing the two criteria at the same time is impossible for general weighted graphs, so we must make a difficult (and somewhat arbitrary) decision about how to trade one for the other. From the practical side, building an MST for general weighted graph in fault-tolerant and self-stabilizing manner is already complex to implement. Finally, typical real networks have a relatively simple structure we can (and should) exploit. That is, nodes close to each other in terms of physical proximity can logically connect to each other at some level and below. Therefore these nodes should be able to form a list entirely within the clique. We have shown this is in fact possible with a very simple hill-climbing with fault-tolerance and adaptiveness.

5.2. Application-Level Multicast and CDN. Our work is in spirit similar to a number of work on application-level multicast and content distribution networks (CDN). Our optimization criteria are different from them, particularly in that we try to reduce the number of branches.

ALMI [9] uses a centralized tree management scheme and makes MST for good performance. End System Multicast [7] takes both latency and bandwidth into account when making a tree of end-hosts. In [12], CAN [11] is used for the infrastructure of multicast. Bayeux [15] uses Tapestry [14] that is also content-addressable network. Overcast [8] is a multicasting system that achieves both small latencies and high throughput. The main application of these systems is multimedia streaming to widely distributed nodes. In such settings, it is important to bound latencies because the application may be an interactive multimedia application. Also in CDNs, the main criteria are latencies and traffic load balancing, rather than delivering as much bandwidth as possible. So researches about CDN such [10, 4, 3] mainly concern how to allocate replicas of contents, and how to redirect user requests to appropriate replicas. On the other hand, it is less important for such applications to squeeze the available bandwidth of local area networks, because there are typically a small number of participating nodes within each network. In contrast, our file replication does not have to optimize

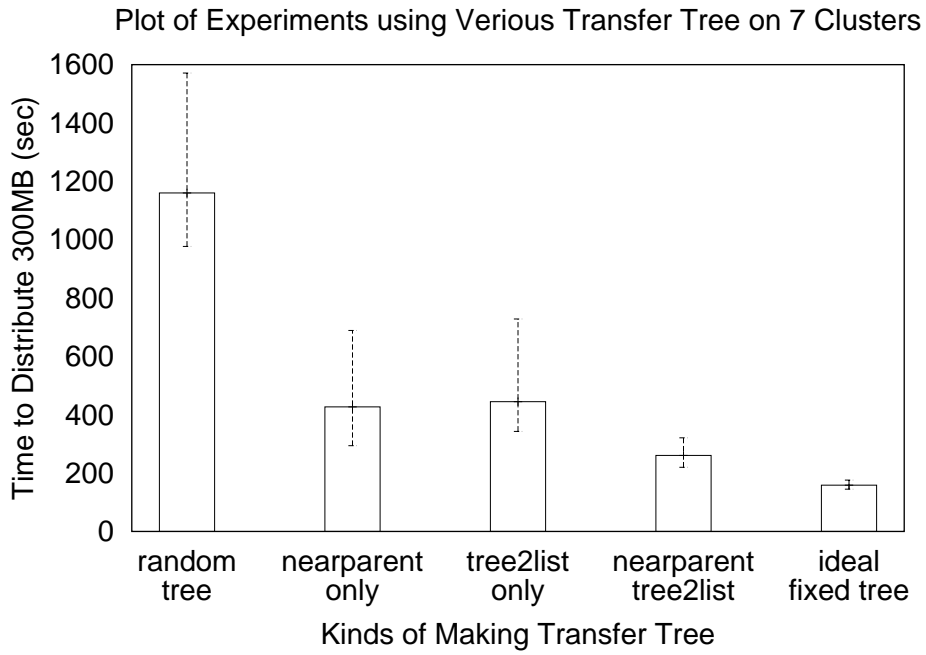


FIG. 4.3. Performance on 7 clusters

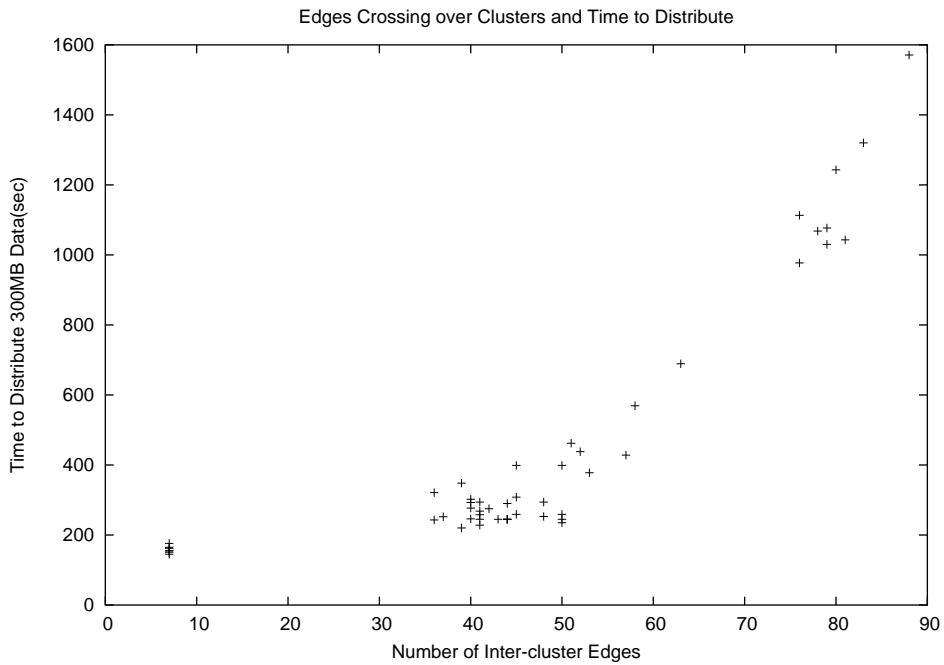


FIG. 4.4. Correlation between number of inter-cluster edges and distribution time

latencies aggressively, because the first priority is on the completion time of transferring large files. It is also very important to utilize LAN bandwidth as much as possible, as the typical usage will be to copy large files to many nodes in clusters. These differences lead them to different optimization criteria, with ours including a unique Tree2List heuristics.

6. Summary and Future Work. We have described a large file distribution algorithm that realizes scalability, adaptiveness, fault-tolerance, and efficient use of bandwidths. It is based on a simple distributed algorithm with simple local heuristics to optimize transfers. We formalized and proved the properties of our algorithm and argued that this gives a good result in practical settings. Our system will be useful for setting up a number of clusters and preparing wide-area distributed computations with a large data. Evaluations show that our implementation is effective in real environment consisting of over 150 nodes across seven clusters campus-wide.

Our current implementation of the protocol is not secure. Any malicious node can participate in the replication and breaks the integrity. To be a useful tool for distributed computing, we must use a suitable authentication when nodes connect to each other. While introducing secure authentications is possible, this may increase the cost of deploying such tools, whose very purpose will be to help maintain a large number of nodes easily. We must study how to maintain ease of installation and use of this tool while achieving a reasonable level of security.

REFERENCES

- [1] Abhishek Agrawal and Henri Casanova. Clustering Hosts in P2P and Global Computing Platforms. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 367–373, 2003.
- [2] F. Bauer and A. Varma. Distributed Algorithms for Multicast Path Setup in Data Networks. Technical Report UCSC-CRL-95-10, University of California at Santa Cruz, August 1995.
- [3] A. Biliris, C. Cranor, F. Douglis, M. Rabinovich, S. Sibal, O. Spatscheck, and W. Sturm. CDN brokering. In *Proceedings of WCW'01*, June 2001.
- [4] Pei Cao and Sandy Irani. Cost-aware WWW proxy caching algorithms. In *Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97)*, Monterey, CA, 1997.
- [5] CVS home. <http://www.cvshome.org/>.
- [6] Lisa Higham and Zhiying Liang. Self-Stabilizing Minimum Spanning Tree Construction on Message-Passing Networks. In *Proceedings of the 15th Conf. on Distributed Computing, DISC, LNCS 2180*, pages 194–208, 2001.
- [7] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture. In *ACM SIGCOMM 2001*, San Diego, CA, August 2001. ACM.
- [8] John Jannotti, David K. Gifford, Kirk L. Johnson, M. Frans Kaashoek, and James W. O'Toole, Jr. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the Fourth Symposium on Operating System Design and Implementation (OSDI)*, pages 197–212, October 2000.
- [9] Dimitris Pendarakis, Sherlia Shi, Dinesh Verma, and Marcel Waldvogel. ALMI: An Application Level Multicast Infrastructure. In *Proceedings of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS '01)*, pages 49–60, San Francisco, CA, USA, March 2001.
- [10] Lili Qiu, Venkata N. Padmanabhan, and Geoffrey M. Voelker. On the placement of web server replicas. In *INFOCOM*, pages 1587–1596, 2001.
- [11] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on applications, technologies, architectures, and protocols for computer communications (SIGCOMM 2001)*, pages 161–172. ACM Press, August 2001.
- [12] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-Level Multicast Using Content-Addressable Networks. *Lecture Notes in Computer Science*, 2233, 2001.
- [13] Yasuhito Takamiya, Atsushi Manabe, and Satoshi Matsuoaka. Lucie: A fast installation and administration tool for large-scaled clusters (in Japanese). In *SACIS 2003*, pages 365–372, May 2003.
- [14] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.
- [15] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 2001)*, June 2001.

Appendix A. Omitted Proofs. In this section we abbreviate is_closer to \mathcal{C} .

A.1. Lemma 3.1. Let V be the set of all nodes. We introduce an unknown x_{AB} for each $A, B \in V$. For each triple (A, B, C) such that $\mathcal{C}(A, B, C)$ is true, we generate a constraint $x_{AB} < x_{AC}$. We then unify x_{AB} and x_{BA} for all $A, B \in V$, replacing all occurrence of one with the other. We are going to show there are no loops of constraints $x_{AB} < x_{CD} < \dots < x_{AB}$, thus the constraints are satisfiable. When we have proved this, we let $d(A, B) = x_{AB}$, for all $A, B \in V$.

To begin with, we show the following:

$$\begin{aligned} x_{AB} &< \dots < x_{YZ} \\ \Rightarrow \mathcal{C}(A, B, Z) \text{ or } \mathcal{C}(A, B, Y), \end{aligned}$$

by induction on the length (the number of inequalities) of the lefthand side n .

1. $n = 1$:

Observe we must have $A = Y$, $A = Z$, $B = Y$, or $B = Z$ since this constraint was generated from \mathcal{C} . When $A = Y$, $x_{AB} < x_{YZ} \Rightarrow x_{AB} < x_{AZ} \Rightarrow \mathcal{C}(A, B, Z)$. Other cases are similar.

2. Assume the claim holds up to $n - 1$ and now we have

$$x_{AB} < x_{CD} < \dots < x_{YZ}$$

of length n . By induction hypothesis, we either have:

- (a) $\mathcal{C}(C, D, Z)$, or
- (b) $\mathcal{C}(C, D, Y)$.

By $x_{AB} < x_{CD}$, we either have:

- (i) $A = C$ and $\mathcal{C}(A, B, D)$,
- (ii) $A = D$ and $\mathcal{C}(A, B, C)$,
- (iii) $B = C$ and $\mathcal{C}(A, B, D)$, or
- (iv) $B = D$ and $\mathcal{C}(A, B, C)$.

Since (a) and (b) are similar we only prove the case (a) by analyzing the four cases (i)–(iv).

- (i) $\mathcal{C}(A, B, D)$ and $\mathcal{C}(A, D, Z)$
 $\Rightarrow \mathcal{C}(A, B, Z)$
- (ii) $\mathcal{C}(A, B, C)$ and $\mathcal{C}(C, A, Z)$
 $\Rightarrow \mathcal{C}(A, B, C)$ and $\mathcal{C}(A, C, Z)$
 $\Rightarrow \mathcal{C}(A, B, Z)$.
- (iii) $\mathcal{C}(A, B, D)$ and $\mathcal{C}(B, D, Z)$
 $\Rightarrow \mathcal{C}(B, A, D)$ and $\mathcal{C}(B, D, Z)$
 $\Rightarrow \mathcal{C}(B, A, Z) \Rightarrow (A, B, Z)$.
- (iv) $\mathcal{C}(A, B, C)$ and $\mathcal{C}(C, B, Z)$
 $\Rightarrow \mathcal{C}(B, A, C)$ and $\mathcal{C}(B, C, Z)$
 $\Rightarrow \mathcal{C}(B, A, Z) \Rightarrow (A, B, Z)$.

Now we prove by contradiction there are no loops:

$$x_{AB} < \dots < x_{YZ} < x_{AB}.$$

By the above induction, we either have:

- (a) $\mathcal{C}(A, B, Z)$ or,
- (b) $\mathcal{C}(A, B, Y)$.

By $x_{YZ} < x_{AB}$, we either have:

- (i) $Y = A$ and $\mathcal{C}(A, Z, B)$,
- (ii) $Y = B$ and $\mathcal{C}(B, Z, A)$,
- (iii) $Z = A$ and $\mathcal{C}(A, Y, B)$, or
- (iv) $Z = B$ and $\mathcal{C}(B, Y, A)$.

We see combining any of (a)–(b) and any of (i)–(iv) will lead to contradiction. We only prove case (a) since (b) is similar.

- (i) $\mathcal{C}(A, B, Z)$ and $\mathcal{C}(A, Z, B)$
 \Rightarrow false.
- (ii) $\mathcal{C}(A, B, Z)$ and $\mathcal{C}(B, Z, A)$
 $\Rightarrow \mathcal{C}(B, A, Z)$ and $\mathcal{C}(B, Z, A)$
 \Rightarrow false.
- (iii) $\mathcal{C}(A, B, Z)$ and $Z = A \Rightarrow$ false.
- (iv) Same as (iii).

A.2. Lemma 3.2. Analyze the three cases, (i) $d(A, C) < d(A, B)$, (ii) $d(A, B) < d(A, C)$, and (iii) $d(A, B) = d(A, C)$. Prove each case by contradiction.

- (i) Let us assume $d(A, C) < d(A, B) < d(B, C)$. Then,
 $d(A, C) < d(A, B)$ and $d(A, B) < d(B, C)$
 $\Rightarrow d(A, C) < d(A, B)$ and $d(B, A) < d(B, C)$
 $\Rightarrow \mathcal{C}(A, C, B)$ and $\mathcal{C}(B, A, C)$
 $\Rightarrow \mathcal{C}(A, C, B)$ and $\mathcal{C}(A, B, C)$
 \Rightarrow false.

- (ii) Similar to (i).
- (iii) Let us assume $d(A, B) = d(A, C) < d(B, C)$. Then,
 - $d(A, B) = d(A, C)$ and $d(A, C) < d(B, C)$
 - $\Rightarrow d(A, B) = d(A, C)$ and $d(C, A) < d(C, B)$
 - $\Rightarrow d(A, B) = d(A, C)$ and $\mathcal{C}(C, A, B)$
 - $\Rightarrow d(A, B) = d(A, C)$ and $\mathcal{C}(A, C, B)$
 - $\Rightarrow d(A, C) = d(A, B)$ and $d(A, C) < d(A, B)$
 - \Rightarrow false.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 3, 2003.

Accepted: September 1, 2003.



NETWORK SCHEDULING FOR COMPUTATIONAL GRID ENVIRONMENTS

MARTIN SWANY* AND RICH WOLSKI†

Abstract.

The problem of data movement is central to distributed computing paradigms like the Grid. While often overlooked, the time to stage data and binaries can be a significant contributor to the wall-clock program execution time in current Grid environments.

This paper describes a simple scheduler for network data movement in Grid systems that can adaptively determine data distribution schedules at runtime on the basis of Network Weather Service (NWS) performance predictions. These schedules take the form of “spanning trees.” The distribution mechanism is an enhancement to the Logistical Session Layer (LSL), a system for optimizing data transfers using “logistics.”

Key words.

Grid computing, data logistics, data staging

1. Introduction. As Computational Grid environments proliferate, the community must constantly evolve the way in which computing systems are used. Distributed computing on the Grid has enabled new ways of harnessing computing resources and yet, has exposed its own set of challenges. One such problem is that of data movement. Applications that are drawn to the Grid because of large resource requirements frequently consume or generate large amounts of data. The problems of data locality and data movement are becoming more prominent and critical to the performance and deployability of Grid systems. Further, due to the dynamism inherent in Grid environments, it is clear that mechanisms for data staging must be adaptive like the computations themselves.

AppLeS [8] demonstrated the beginning of a new way of thinking about programming the Grid—scheduling from the perspective of the application. In this spirit, we propose to approach the problem of adaptively scheduling buffers in the network with proactive support from the application. This paper examines simple optimizations that we can facilitate by thinking of Grid resources in terms of cooperating elements in a storage and computing “overlay” network. By enabling this type of functionality, using techniques such as the Logistical Session Layer (LSL) [34] or the Internet Backplane Protocol (IBP) [28], the breadth of the services offered by a Grid is improved.

The goal of this work is to investigate scheduling and routing techniques focused on optimizing data movement in Grid environments. In order to investigate such scheduling we will draw on previous work as follows. The Logistical Session Layer (LSL) [34] provides the basic platform for cooperative data forwarding that responds to requests from the scheduler. The Network Weather Service (NWS) [43] provides us with network performance monitoring and forecasting capabilities. Finally, the NWSlapd [37], the caching and delivery subsystem of the NWS, caches network performance forecasts and aggregates them into a form suitable for consumption by the scheduler.

There has been a tremendous amount of work in this community to optimize collective operations for parallel computing [4, 27, 24, 5, 18, 39, 20, 40]. Certainly, these approaches are all related at some fundamental level (and discussed somewhat in Section 6). However, our approach is focused on pre-runtime data distribution (or staging) rather than collective operations as such. Initial data distribution is an important component of actual Grid deployment. This fact is often obscured by pre-staged binaries or locally-generated random input data, but for Grid systems to realize their potential, these issues must be addressed.

Our approach to this problem is unique in a number of key ways:

- It treats Grid resources as a graph with edge values derived from current network performance forecasts
- It adaptively builds distribution trees for arbitrary topologies by creating a schedule based on the Minimum Spanning Tree (MST) over that graph
- Cooperative forwarding among peers is accomplished with the *Logistical Session Layer* (LSL), which uses cascaded TCP connections.

Grid environments are extremely dynamic. Network performance depends on ambient load. To best adapt our execution at runtime, forecasts based on current performance information are necessary. Distribution trees based on this information will often vary wildly in shape. We need an extremely general tree construction mechanism to accommodate the diversity of Grid systems. Finally, as we use LSL for our distribution platform,

*Department of Computer and Information Sciences, University of Delaware, Newark, DE, 19716 (swany@cis.udel.edu).

†Department of Computer Science, University of California, Santa Barbara, CA, 93106 (rich@cs.ucsb.edu).

we get the benefits of performance-enhancing buffering in the network, and the reliability and deployability of TCP.

In this paper we will first describe the assumptions in our approach to scheduling. Next, we will describe a simple scheduling approach, based on spanning tree, that is general enough to address our needs. Finally, we describe the enhancements to LSL necessary to implement a schedulable distribution mechanism and evaluate the performance improvements that even simple scheduling can afford in this space.

2. Problem. The general problem that this work addresses is that of the “logistics” of data movement in Computational Grid environments. In fact, the logistics of data movement are the main reason why computing “power” is not a fungible resource like electrical power. Users need computations to be performed on specific bits of data, whereas electricity can be consumed regardless of the location or means of its generation. The problems of data locality and movement are universal and are a critical consideration in Grid systems.

There has been much recent work considering cooperative data sharing between networked peers [30, 33, 6, 28, 22]. These cooperative approaches have had impact in both the parallel processing and network computing domains. In this spirit, we consider an environment in which Grid resources are enabled to utilize and provide cooperation of this sort. Our goal is to consider scheduling these resources and examine potential performance optimizations that might emerge. This work builds on the ideas of “Logistical” [34, 6, 28], “overlay” [3, 38, 17] and “peer-to-peer” [30, 33, 22, 44] networking to treat the problems of communication in Grid systems in a novel manner.

The GrADS [7] project is a large, multi-institution project whose goal is to investigate comprehensive software environments for developing Grid applications. As such, the GrADS environment is focused on program development and compilation as well as runtime Grid support. Before execution, a Configurable Object Program is prepared by the compilation systems. When the program is to be launched, the Scheduler/Service Negotiator (S/SN) interacts with a variety of runtime services provided by the Grid fabric and discovers the “state” of the Grid at that time. The S/SN uses this state information to make decisions about program configuration and scheduling. In particular, the system requires current short-term forecasts of resource performance levels so that it can make proactive scheduling decisions. The NWS generates such forecasts automatically, but to be useful, they have to be delivered to the S/SN (through the Globus [13] infrastructure) quickly and reliably.

Considering the problem of initial data distribution, our assumptions can be captured by the following scenario. Let us imagine that a user is launching a program in a Grid environment such as the GrADS [7] project’s testbed. In the GrADS architecture, the Configurable Object Program, or COP, is distributed by the Application Manager in the first phases of execution. This is not, of course, unique to GrADS. In many Grid paradigms a user has a set of program executables that need to be distributed to the resources before execution can begin.

In other Grid usage models, end-users utilize resources through previously existing software infrastructure. This software exports services through application interfaces using remote procedure calls, or RPC. NetSolve [11] is an example of such a system. The problem that these systems face is similar to the program distribution problem in that some amount of data must often be sent from the user to Grid resources prior to the beginning of any meaningful execution. This problem is strongly related in that it concerns initial data distribution and thus, it can be modeled similarly.

These problems are equivalent to some degree in that either prior to runtime or during an initial phase of runtime, some data has to be sent to the each computational node before any real application progress can be made. Often, we choose to abstract this problem away with file-sharing techniques. In fact, network file systems (e.g. NFS) can be used within a single site so that we only need to transfer once to nodes that share files this way, but there are many cases where systems do not share files in this fashion. Further, NFS can suffer from poor performance and since data (programs or user data) is to be moved over the network, we prefer to deal with the associated overhead explicitly. Certainly, there are many situations and scenarios that differ in simple ways from this basic model, but this captures our assumptions and, in fact, models real Grid systems quite well.

2.1. Problem Modeling. Consider the simple depiction of these data transfers in Figure 2.1. In these graphs, the value along the edge denotes some cost. In this case it is the time to transfer some amount of data. Figure 2.2 obviously demonstrates a distribution pattern (or tree) with a lower overall cost.

Further, in Grid environments, resources are often located in groups or clusters, so the potential performance improvement from such optimizations becomes more obvious. Figure 2.3 illustrates the fact that in many real cases, a hierarchical distribution scheme can greatly reduce the overall cost of the paths through the network.

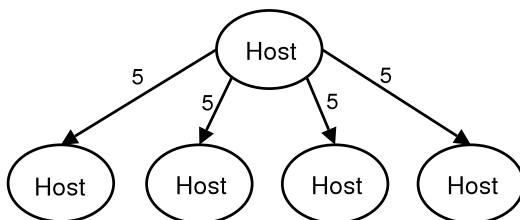


FIG. 2.1. Cost tree for default distribution strategy

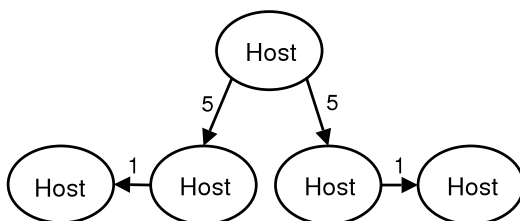


FIG. 2.2. Less costly distribution tree

This modeling approach allows us to think about the problem of data distribution as a graph and offers obvious chances for optimization.

3. Scheduling Algorithms. The crux of this work is the observation that by treating the resources of the Grid as a “network”, we can schedule the cooperation of these resources in the formation of a single-source, data distribution tree. This schedule can be computed dynamically, based on current performance information. A distribution tree must be able to direct the data to each node, or “span” the tree.

Consider a directed graph G with vertices and edges: $G = (V, E)$. Each edge has a weight or cost c_{ij} for each $(i, j) \in E$. A spanning tree (T) is a graph with $T \subseteq G$ such that $\forall V$ there is a $(u, v) \in T$ that is incident on it (i.e., T spans the set V).

The Minimum Spanning Tree $MST(G) = T$ where $\sum_{(u,v) \in T} c(u, v)$ has the minimum cost of all spanning trees.

A traditional, and provably optimal, approach to the solution of MST is known as Prim’s algorithm [29]. This algorithm uses a greedy approach in the construction of the solution tree. Briefly, the algorithm proceeds as follows.

To find the MST (T), we create an empty tree T and move the starting node of the tree (v_{start}) from V to T :

$$v_{start} \in T \mid T \cap G = \emptyset \quad (3.1)$$

Then, we iterate while $|V| > 0$. At each step we examine edges in the “cut” (edges that begin in T and end in V) and select the minimum cost edge:

$$\min(e) \in E' \mid e(u, v) \ u \in T \text{ and } v \in V \quad (3.2)$$

Node v is then moved to T and we examine the newly added node and edge to see if its addition has offered a better path to nodes already in T .

While the spanning tree problem is at the heart of this approach to scheduling, there are additional factors that must be considered in our model. In the previous section, we considered extremely simple graphs. Obviously for Internet hosts, the time to transmit data to a number of hosts is not linear with the number of hosts. Multiple outgoing edges interfere with one another – they are not independent. In terms of the network, the more streams there are sharing the resource of outgoing network capacity, the less each stream gets. This could complicate the model significantly. In fact this problem is very similar to what is known as the “weighted graph minimum-energy broadcast problem”, which has been shown to be NP-hard [41]. Further work in the same problem

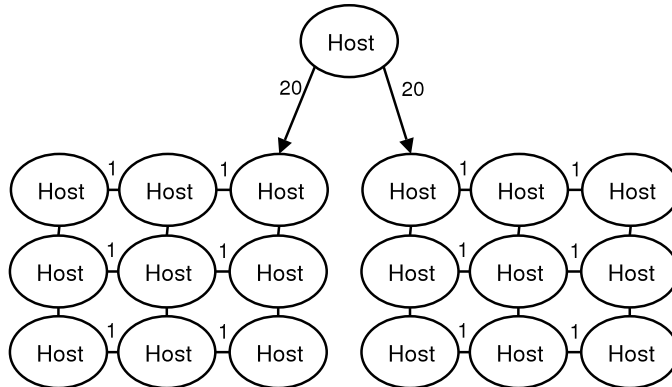


FIG. 2.3. A distribution tree for clusters

space [12] shows that the problem remains NP-hard even when realistic bounds are placed on transmission levels (reducing them to a small fixed set), but gives hope for polynomial-time solutions if a solution exists.

Another potential combinatorial problem arises in our situation as well. The “Steiner Network” is different from the MST problem in that only a subset S of G must be spanned. This problem has been shown to be NP-hard [19]. This problem is the heart of the problem of “minimum spanners” [10] again demonstrated to be NP-complete. However, we note that since the set with which we are concerned is not a subset of G that we avoid the difficulties associated with these problems.

These previous results treat their realm of discourse to be in “metric space,” meaning that the triangle inequality holds. Internets are not, in general, in metric space. This makes the problem more tractable initially, but ultimately complicates the model. In particular, rather than power levels, our spanning-tree problem has the above described constraint that we can refer to as “lateral inhibition.” The more edges (streams) that are incident on a node, the less well any of them perform. In the extreme, the interference between streams is unique for every stream configuration. This combinatorial space implies that the optimal solution for such a problem is NP-hard. However, we note that this approach is not necessarily concerned with an optimal solution, rather we wish to empirically determine the efficacy of this general class of solution.

The MST problem is known to be related to many problems in distributed data movement. While we do not deal with it directly in this work, the minimum cost path and all-pairs minimax problems [2] provide a basis for multi-hop forwarding of the sort proposed by LSL [34] and IBP [28]. Parallel streams with diverse paths allow us to couch routing in terms of maximum flow algorithms. However, utilizing parallel streams between identical locations, with default paths, only serves to increase the value of a single arc. This would certainly increase the observed bandwidth, but our treatment of the single-stream case still holds without loss of generality.

4. System Architecture. To deploy and test this scheduler on a Grid system, we rely on various components of Grid software. Specifically, this software depends on the Network Weather Service, the NWS’s caching LDAP delivery system and the Logistical Session Layer.

4.1. Network Weather Service. The Network Weather Service [43, 42] is a system developed to provide performance monitoring and online performance prediction to Grid schedulers such as ours. Grid environments are extremely dynamic and in order to manage this dynamism, a scheduler must have near-term performance predictions upon which to base runtime decisions. The NWS measures, among other things, TCP bandwidth and latency between hosts in a scalable and unintrusive manner. By applying various non-parametric statistical techniques on the timeseries produced by these ongoing measurements, the NWS is able to produce forecasts that greatly improve prediction over naive techniques. Further, these measurements can be combined with past instrumentation data to produce accurate estimates of bandwidth [36] or transfer time.

An additional component of the NWS, called the NWSlapd [37, 35], provides necessary functionality as well. First, this system caches performance predictions near querying entities making it possible to scale the performance information infrastructure and provide ubiquitous forecasts to network-aware schedulers. This part of the system also assembles measurement information into a network “view” that can be easily and quickly queried. Note, however, that the NWS does not actually initiate measurements between every pair of hosts (n^2

tests.) Rather, the NWSlapd interprets the hierarchy of measurements that the NWS does take and fills in a complete matrix of forecasts (as described in [35].)

The complete matrix of forecasts provides us with the node-node adjacency matrix representation of our network. The adjacency matrix is populated by the observed bandwidth (and/or latency) between host i and host j in the (i, j) th element. Note that the graph that this matrix represents is fully-connected as every host on the Internet can reach every other host with some bandwidth.¹ This provides the initial graph G upon which our scheduler operates.

4.2. Scheduler Implementation. Our initial scheduling approach is simply to describe a spanning tree for the nodes in our resource pool. To do this, we simply use Prim’s algorithm as described in Section 3. In order to produce a minimum spanning tree, we need a metric where a smaller value is “better”. Since we are operating with bandwidth forecasts, we convert the bandwidth estimates “transfer time” estimates by considering $1/\text{bandwidth}$ as the “value” of an edge.

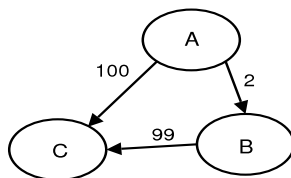


FIG. 4.1. *Simple Illustration of Tree Depth*

One simple technique that we have implemented allows us to minimize the depth of the spanning tree. Our goal is to minimize the number of hops that a stream must pass through as each hop adds some amount of overhead. Consider the graph in Figure 4.1. Strictly speaking, the minimum spanning should include the arc $A \rightarrow B$, and that from $B \rightarrow C$. However, it reduces the depth by a level and increases the overall cost of the tree to span via the arc from $A \rightarrow C$.

This has an effect in practice. Due to small variations in measurements through time, machines with functionally similar connectivity have slightly different forecasts. To keep the trees more simple, we would like to consider measurements within some ϵ of one another as the same. A perfect choice for this value is the historical forecasting error from the NWS.

The scheduler performs as expected. When presented with the results of a performance query from NWS containing information about the GrADS testbed [14], the system was clearly able to discern separate clusters at the University of Tennessee and University of Illinois and suggest a distribution tree taking that into account. Figure 4.2 depicts spanning tree produced by the scheduler, and this graph is generated from that output using GraphViz [15], a graph plotter. The initial set of results (in Section 5) utilize this host pool and similar distribution schedules.

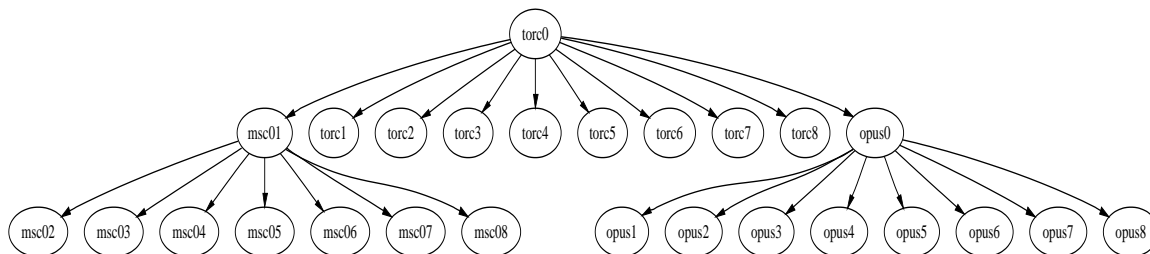


FIG. 4.2. *Spanning Tree*

Note that Figure 4.2 is created automatically. Other than guessing based on the names of the hosts (not on the domain name), there is no way to discern these clusters at the network level. In some cases, only empirical performance measurements show these relationships, as shown previously by Effective Network Views

¹With the exception of hosts behind firewalls. While our techniques are even more natural in those cases, a discussion of that application beyond the scope of this work.

(ENV) [32]. It is interesting to note that we have recovered the structure of the network with our scheduler technique alone.

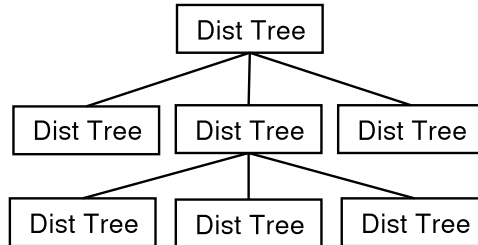


FIG. 4.3. *Distribution records in a tree*

4.3. Logistical Session Layer Data Distribution. The scheduler produces a distribution tree which is given to the Logistical Session Layer [34] (LSL) to control the data distribution. LSL is a system for cooperative forwarding and buffering of network traffic that has been shown to greatly increase end-to-end network performance. LSL utilizes TCP, so questions of “friendliness” are not an issue and data integrity guarantees are those of TCP.² However, LSL endeavors to allow TCP to perform better by keeping the round-trip time on any sublink to a minimum. This use of TCP also facilitates incremental deployability, yet takes advantage of improving transport-layer performance.

For this particular experiment, we have implemented a new message option in the LSL stack. Each option defines a distribution tree including information about the children of that node. The hierarchy of distribution headers is recursively encoded and decoded so that only the relevant portions of the subtree are transmitted to downstream neighbors until ultimately, the leaf nodes get a distribution tree with a single entry. Figure 4.3 illustrates this.

The acknowledgment of data receipt at the ultimate destination is implicit with the closing of the TCP socket. At each LSL node, necessary data is sent out all outgoing sockets and the sending side of each of those sockets is closed. Each daemon then waits for each downstream neighbor to close its socket, signaling that all destinations have received the data. At the leaf nodes, the sockets are closed normally once all data is written to the filesystem. We note that direct notification from destination to source may be more desirable in many cases and such a modification is straightforward.

Internally, the implementation is not aggressively optimized, and further performance improvements are certainly possible. There is also no security model at this time. Our technique could easily work over SSH-encrypted and authenticated tunnels and this is one implementation possibility that we are investigating.

5. Results. To test the efficacy of our system, we have deployed it across the GrADS testbed [14]. This set of Grid resources ranges from 50 to 100 nodes across the U.S. located primarily at the University of California, San Diego, the University of Illinois, Champaign-Urbana, and the University of Tennessee, Knoxville. The sites are connected by Internet2’s Abilene [1] backbone and enjoy relatively high-speed connectivity.

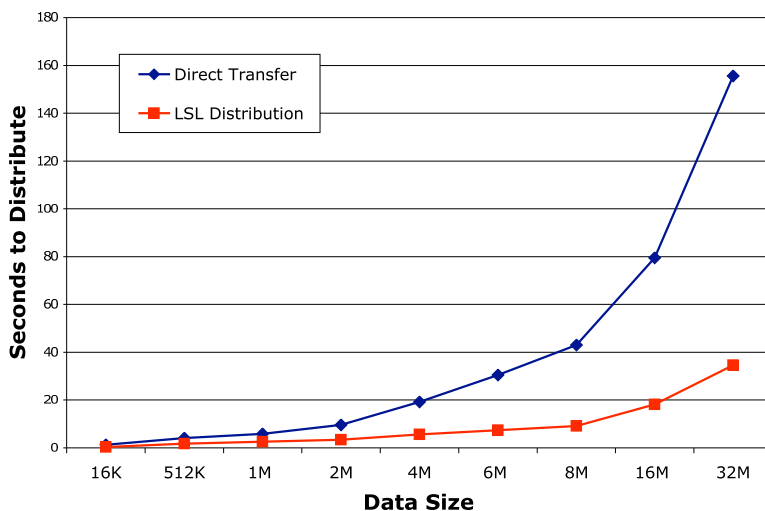
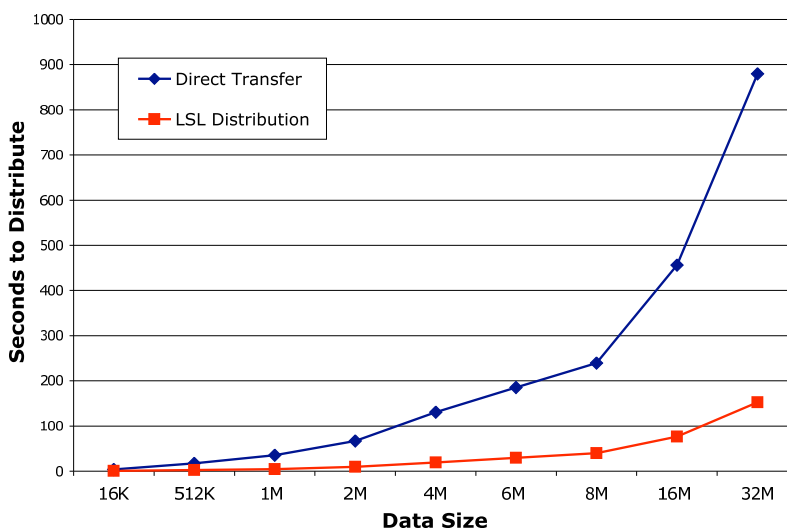
To evaluate the difference between direct distribution (the direct approach) and our scheduler in as fair a manner as possible, we have modeled the direct distribution within our software infrastructure. That is, the direct distribution version is simply a flat tree. This allows for overlapping communication among the streams and is not terribly inefficient. At any rate, the data movement is not serialized among the nodes as it often is in daily use.³

Two sets of tests were run. The first set contains 18 nodes located at two sites. The second set contains 52 nodes in 6 clusters at 3 sites. In all cases the source of the data was located at the University of California, Santa Barbara. Again, this models situations that are demonstrably realistic.

Figure 5.1 shows the distribution time, in seconds, for files of various sizes. This test utilized the 18 node pool described above. We can see that this case illustrates remarkably well how hierarchical, cooperative data distribution can improve performance and reduce distribution time. Figure 5.2 shows file distribution times for the larger (52 node) host pool. Again, the performance improvement from making simple scheduling decisions

²Whether this is sufficient or not is another matter, as we have done no harm.

³The authors speak from experience. What Grid developer hasn’t iterated through a file copy to each node of some set?

FIG. 5.1. *Distribution Times for 18 Hosts*FIG. 5.2. *Distribution Times for 52 Hosts*

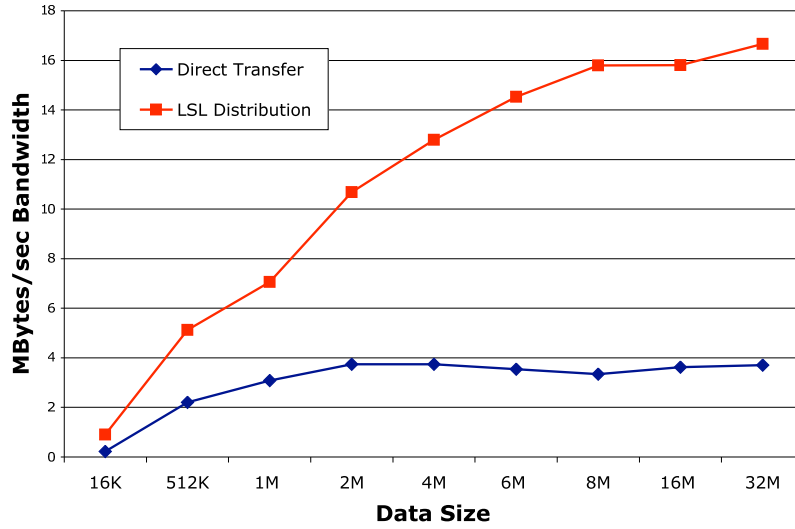
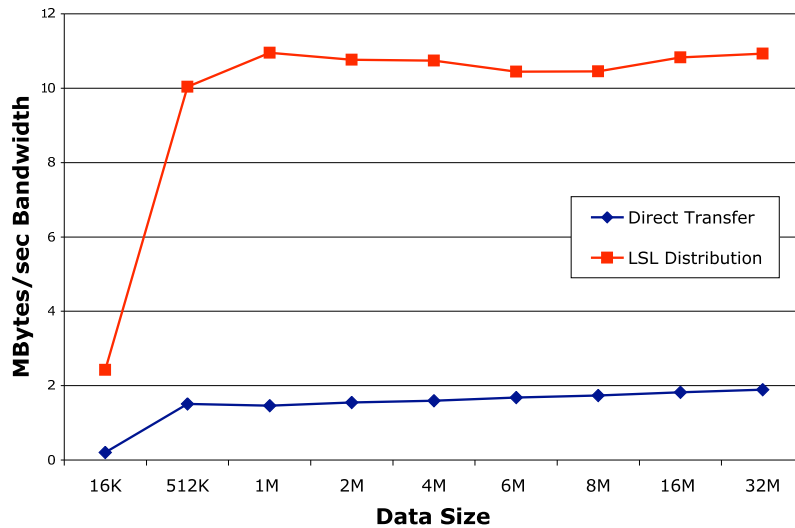
is quite significant. We note that clusters represent the best case for distribution techniques such as this and clusters are frequently components in a Grid.

Figure 5.3 depicts the delivered bandwidth that we observe in data transfers to the 18 node host pool. Figure 5.4 shows this same metric for the larger host pool. We have initiated a data transfer that has an average performance more than the physical link to which the machine is attached (12.5MB/sec).

6. Related Work. There are many aspects of research that are similar and related. LSL is part of the more general inquiry of Logistical Networking [28, 6]. This work investigates a more rich view of storage in the network and our scheduling approach is applicable to either infrastructure.

Globus GASS [9] and GridFTP [16] are data movement and staging service for Grid systems that could be scheduled using the techniques that we have described. The MagPIe [20, 40, 21] project has investigated performance optimizations for collective operations. Improving the performance of collective operation has been investigated in many different contexts [4, 27, 24, 5, 18, 39], although primarily the focus has been MPI.

Scheduling application activity based on the state of the network is seen many places including REMOS [23], Topology-d [26] and the Network Weather Service [42].

FIG. 5.3. *Delivered Bandwidth of Distribution Tree (18 Hosts)*FIG. 5.4. *Delivered Bandwidth of Distribution Tree (52 Hosts)*

Our approach is quite similar to recent work by Malouch, et. al [25], which treats multicast proxies as nodes in a network optimization problem. We note that their arc incidence constraints are different than those that we propose. Further, their simulations were aimed at evaluating various heuristics, while our goal is to understand the performance improvements from simple scheduling in real networks.

Overcast [17] is a network overlay based multicast system. Overcast uses node to node protocols to build and evaluate the distribution trees. Our approach creates distribution trees at runtime and assumes no state in the network. Rather, we assume the availability of network performance forecasts to determine distribution trees. Our concerns about node failure are also quite different given our utilization of TCP as a transport layer.

Recent work in application-level multicast explores the applicability of peer-to-peer networks [31] for this purpose. They note a benefit of their work is the lack of a constantly-running routing protocol, a benefit that we share. In contrast to their approach, however, we don't increase the time to distribute data, rather we decrease it.

7. Conclusion. We have focused on the problem of initial data distribution in Grid environments. By building on previous system components, such as the NWS and LSL, we have developed a novel system for data distribution. We have developed a scheduler that is able to instantiate cooperative data forwarding based on LSL and performance data from NWS. This scheduling technique and infrastructure allow us to form distribution trees that greatly increase performance and reduce time to distribute data. Techniques such as this will only become more important as Grids proliferate.

REFERENCES

- [1] *Abilene*, <http://www.ucaid.edu/abilene/>.
- [2] R. AHUJA, T. MAGNANTI, AND J. ORLIN, *Network Flows: Theory, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, New Jersey, 1993.
- [3] D. ANDERSEN, H. BALAKRISHNAN, M. KAASHOEK, AND R. MORRIS, *Resilient overlay networks*. Proc. 18th ACM SOSP, Banff, Canada, October 2001.
- [4] V. BALA, J. BRUCK, R. CYPHER, P. ELUSTONDO, A. HO, C.-T. HO, S. KIPNIS, AND M. SNIR, *CCL: A portable and tunable collective communication library for scalable parallel computers*, IEEE Transactions on Parallel and Distributed Systems, 6 (1995), pp. 154–164.
- [5] M. BANIKAZEMI, V. MOORTHY, AND D. PANDA, *Efficient collective communication on heterogeneous networks of workstations*, in International Conference on Parallel Processing, 1998, pp. 460–467.
- [6] M. BECK, T. MOORE, J. PLANK, AND M. SWANY, *Logistical networking: Sharing more than the wires*, in Proc. of 2nd Annual Workshop on Active Middleware Services, August 2000.
- [7] F. BERMAN, A. CHIEN, K. COOPER, J. DONGARRA, I. FOSTER, L. J. DENNIS GANNON, K. KENNEDY, C. KESSELMAN, D. REED, L. TORCZON, , AND R. WOLSKI, *The GrADS project: Software support for high-level grid application development*, Tech. Report Rice COMPTR00-355, Rice University, February 2000.
- [8] F. BERMAN, R. WOLSKI, S. FIGUEIRA, J. SCHOPF, AND G. SHAO, *Application level scheduling on distributed heterogeneous networks*, in Proceedings of Supercomputing 1996, 1996.
- [9] J. BESTER, I. FOSTER, C. KESSELMAN, J. TEDESCO, AND S. TUECKE, *GASS: A data movement and access service for wide area computing systems*, Sixth Workshop on I/O in Parallel and Distributed Systems, (1999).
- [10] L. CAI, *Np-completeness of minimum spanner problem*, Discrete Applied Mathematics, 48 (1994), pp. 187–194.
- [11] H. CASANOVA AND J. DONGARRA, *NetSolve: A Network Server for Solving Computational Science Problems*, The International Journal of Supercomputer Applications and High Performance Computing, (1997).
- [12] O. EGECIOGLU AND T. GONZALEZ, *Minimum-energy broadcast in simple graphs with limited node power*, in Proc. IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2001), August 2001, pp. 334–338.
- [13] I. FOSTER AND C. KESSELMAN, *Globus: A metacomputing infrastructure toolkit*, International Journal of Supercomputer Applications, (1997).
- [14] GRADS, <http://hipersoft.cs.rice.edu/grads>.
- [15] *Graphviz*, <http://www.research.att.com/sw/tools/graphviz>.
- [16] *GridFTP*, <http://www.globus.org/datagrid/gridftp.html>.
- [17] J. JANNOTTI, D. K. GIFFORD, K. L. JOHNSON, M. F. KAASHOEK, AND J. W. O'TOOLE, JR., *Overcast: Reliable multicasting with an overlay network*, in Proceedings of Fourth Symposium on Operating System Design and Implementation (OSDI), October 2000, pp. 197–212.
- [18] N. T. KARONIS, B. R. DE SUPINSKI, I. FOSTER, W. GROPP, E. LUSK, AND J. BRESNAHAN, *Exploiting hierarchy in parallel computer networks to optimize collective operation performance*, in 14th International Parallel and Distributed Processing Symposium, 2000, pp. 377–386.
- [19] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. Miller and J. Thatcher, eds., Plenum Press, 1972, pp. 85–104.
- [20] T. KIELMANN, H. E. BAL, S. GORLATCH, K. VERSTOEP, AND R. F. HOFMAN, *Network performance-aware collective communication for clustered wide area systems*, Parallel Computing, 27 (2001), pp. 1431–1456.
- [21] T. KIELMANN, R. HOFMAN, H. BAL, A. PLAAT, AND R. BHOEDJANG, *Mpi's reduction operations in clustered wide area systems*, 1999.
- [22] J. KUBIATOWICZ, D. BINDEL, Y. CHEN, P. EATON, D. GEELS, R. GUMMADI, S. RHEA, H. WEATHERSPOON, W. WEIMER, C. WELLS, AND B. ZHAO, *Oceanstore: An architecture for global-scale persistent storage*, in Proceedings of ACM ASPLOS, ACM, November 2000.
- [23] B. LOWECAMP, N. MILLER, D. SUTHERLAND, T. GROSS, P. STEENKISTE, AND J. SUBHLOK, *A resource query interface for network-aware applications*, in Proc. 7th IEEE Symp. on High Performance Distributed Computing, August 1998.
- [24] B. LOWECAMP AND A. BEGUELIN, *Eco: Efficient collective operations for communication on heterogeneous networks*. In International Parallel Processing Symposium, pages 399–405, Honolulu, HI, 1996., 1996.
- [25] N. MALOUCH, Z. LIU, D. RUBENSTEIN, AND S. SAHU, *A graph theoretic approach to bounding delay in proxy-assisted*. In 12th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'02), May 2002. 143, 2002.
- [26] K. OBRACZKA AND G. GHEORGHIU, *The performance of a service for network-aware applications*, in Proceedings of 2nd SIGMETRICS Conference on Parallel and Distributed Tools, August 1998.
- [27] J.-Y. L. PARK, H.-A. CHOI, N. NUPAIROJ, AND L. M. NI, *Construction of optimal multicast trees based on the parameterized communication model*, in Proceedings of the International Conference on Parallel Processing (ICPP), 1996, pp. 180–187.
- [28] J. S. PLANK, A. BASSI, M. BECK, T. MOORE, D. M. SWANY, AND R. WOLSKI, *Managing data storage in the network*, IEEE Internet Computing, 5 (2001), pp. 50–58.

- [29] R. PRIM, *Shortest connection networks and some generalizations*. Bell System Technical Journal, 36, 1389–1401, 1957.
- [30] S. RATNASAMY, P. FRANCIS, M. HANDLEY, R. KARP, AND S. SHENKER, *A scalable content-addressable network*, in SIGCOMM, 2001, pp. 161–171.
- [31] S. RATNASAMY, M. HANDLEY, R. KARP, AND S. SHENKER, *Application-level multicast using content-addressable networks*, Lecture Notes in Computer Science, 2233 (2001), pp. 14–25.
- [32] G. SHAO, F. BERMAN, AND R. WOLSKI, *Using effective network views to promote distributed application performance*. In Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications, 1999.
- [33] I. STOICA, R. MORRIS, D. KARGER, F. KAASHOEK, AND H. BALAKRISHNAN, *Chord: A scalable content-addressable network*, in SIGCOMM, August 2001.
- [34] M. SWANY AND R. WOLSKI, *Data logistics in network computing: The Logistical Session Layer*, in IEEE Network Computing and Applications, October 2001.
- [35] ———, *Building performance topologies for computational grids*. Proceedings of Los Alamos Computer Science Institute (LACSI) Symposium, October 2002.
- [36] ———, *Multivariate resource performance forecasting in the network weather service*, in Proceedings of SC 2002, November 2002.
- [37] ———, *Representing dynamic performance information in grid environments with the network weather service*. 2nd IEEE International Symposium on Cluster Computing and the Grid, May 2002.
- [38] J. TOUCH, *The XBone*. Workshop on Research Directions for the Next Generation Internet, May 1997.
- [39] S. VADHIYAR, G. FAGG, AND J. DONGARRA, *Performance modeling for self adapting collective communications for MPI*. Proceedings of Los Alamos Computer Science Institute (LACSI) Symposium, October 2001.
- [40] R. V. VAN NIEUWPOORT, T. KIELMANN, AND H. E. BAL, *Efficient load balancing for wide-area divide-and-conquer applications*, in PPoPP '01: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, June 2001, pp. 34–43.
- [41] P.-J. WAN, G. CALINESCU, X. LI, AND O. FRIEDER, *Minimum-energy broadcast routing in static ad hoc wireless networks*, in INFOCOM, 2001, pp. 1162–1171.
- [42] R. WOLSKI, *Dynamically forecasting network performance using the network weather service*, Cluster Computing, (1998). also available from <http://www.cs.utk.edu/~rich/publications/nws-tr.ps.gz>.
- [43] R. WOLSKI, N. SPRING, AND J. HAYES, *The network weather service: A distributed resource performance forecasting service for metacomputing*, Future Generation Computer Systems, (1999).
- [44] B. Y. ZHAO, J. D. KUBIATOWICZ, AND A. D. JOSEPH, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Tech. Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 5, 2003.

Accepted: September 1, 2003.



TOWARD REPUTABLE GRIDS

G. VON LASZEWSKI*, B. K. ALUNKAL†, AND I. VELJKOVIC‡

Key words. Grid, Quality-of-service, Trust, Reputation

Abstract.

The Grid approach provides a vision to access, use, and manage heterogeneous resources in virtual organizations across multiple domains and organizations. This paper foremost analyses some of the issues related to establishing trust and reputation in a Grid. Integrating reputation into quality management provides a way to reevaluate resource selection and service level agreement mechanisms. We introduce a reputation management framework for Grids to work toward facilitating the complex task of improving the quality of resource selection. Based on community experience we adapt trust and reputation of entities through specialized services. Simple contextual quality statements are evaluated in order to effect the reputation for a monitored resource. Additionally, we introduce a novel algorithm for evaluating Grid reputation by combining two known concepts using eigenvectors to compute reputation and integrating global trust.

1. Introduction. The Grid approach [18, 21] provides a *vision* to develop an environment for coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations under quality-of-service constraints [5, 10]. However, optimal use of these distributed services and resources requires not only knowledge about the capabilities of the resources, but also the assurance that the available and requested capabilities can be used successfully. Grid users are faced with questions such as which resources are available remotely, which capabilities these resources have, whether one is authorized to use these resources, whether the information for a resource selection is accurate, and on which resources a task is likely to execute with the most success.

In a typical Grid scenario users identify possible candidate resources through metainformation obtained from directories, databases, or registries. However, the current generation of Grid information services provides only the most elementary information to guide quality-of-service based resource selection. For example, the Globus Toolkit Monitoring and Directory Service (MDS) [19] provides a limited set of information about Grid resources, including static and possibly dynamic attributes and properties. In many cases the information returned by this service is costly to obtain, inaccurate, or outdated and does not integrate a resource selection service. We observe that, similar to Heisenbergs uncertainty principle [13], the more variability (momentum), the information in regards to a resource attribute contains, the less we can predict the accuracy of its value at a time, and vice versa. This principle is of especial importance if we consider the use of multiple resources in a coordinated fashion, multiplying this effect. Furthermore, the sporadic nature of the Grid and its measured values as well as the possibility of integrating ad hoc services [21] in a Grid environment for which no historical data is available, poses a severe limitation on the current generation of prediction services. Additionally, we often lack information provided on the quality of the participating entities, similar to an Internet shopping site, which classifies included items while augmenting them with information not only about functionality, appearance, availability, and price, but also about appreciations and ratings by its shoppers.

In our framework we propose a probabilistic preselection of resources based on likelihood to deliver the requested capability and capacity. Such a service can be integrated into a quality-of-service management framework [7] to enable the reevaluation of the effectiveness of quality-of-service policies and service level agreements.

This motivated us to design a reputation framework for Grids to assist in the selection process for resources while integrating the notions of trust and reputation. Trust is already a critical parameter in the decision-making process of several peer-to-peer (P2P) frameworks. Reputation is computed by using a trust rating provided by users of services through a feedback mechanism. Reputation-based service and product selection have proved to be a great asset for online sites such as eBay [9] and Amazon [3].

Hence, we propose a framework that selects through a hierarchical process, with the help of sophisticated Grid service, sets of resources and services as suitable candidates to fulfill quality-of-service requirements. This includes the selection of trusted resources that best satisfies application requirements according to a predefined

*Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439, U.S.A. gregor@mcs.anl.gov

†Department of Computer Science, Illinois Institute of Technology, Chicago, IL 60616, U.S.A.

‡Department of Computer Science and Engineering, The Pennsylvania State University, PA 16802, U.S.A.

trust metric. Therefore, we propose that our hierarchical resource selection process be augmented by qualitative and quantitative experiences based on previous transactions with resources so we can integrate this experience in future resource selections.

We envision such a reputation system for Grids, in which resources and services are ranked based on the reputation they obtain. Generating a reputation or establishing trust by entities (resources, services, and individuals) in regards to their availability and capability. We believe that such a reputation service framework is of crucial importance for Grid computing to increase reliability, use, and popularity. Trust and reputation serve as an important metric to avert the use of underprovisioned and malicious resources; they provide the ability to simplify the selection process while focusing first on qualitative concerns.

Consider a Grid environment that agglomerates expensive and specialized resources including high-performance servers, storage databases, advanced scientific instruments, and sophisticated services to visualize macromolecules [22] or nanomaterial [4] structures. Such an environment requires reliable ad hoc Grid services to fulfill the necessary quality-of-service required by secure real-time use. Furthermore, the sporadic and time-limited nature of the services and resources used may result in a lack of historical data, posing severe limitations on existing prediction services.

Community-based adaptive metrics such as trust and reputation serve as building blocks to support our quality-of-service requirements. We emphasize that the self-evaluation of a service must be an integral part of the Grid architecture in order to increase reliability and predictability. Consider the case in which a service claims it will provide a particular level of quality and engages in a service level agreement with another service. Assume that this service fails to deliver the promised agreement. Such a scenario might exist when the metrics available for selection do not coincide with the goals. Choosing a more reliable service can avoid this problem. But how do we know that another service is more reliable?

Concretely, if we try to transfer 10 Gbytes of data between remote resources through a network, we might be tempted to select a network path with the highest observed peak throughput. However, if the network gets interrupted and the transfer would fail, the measurement and metric must take this into account. We cannot rely on a service that selects the route for transfer based only on a simple bandwidth measurement. Rather, we require a service that evaluates the promised agreement and is available for future reference. Hence, we are not only concerned with the quality-of-service, but also with the quality-of-information [20] to establish such a service.

We need to address in an effective quality-of-service framework the following issues:

1. Identify the metrics that are defining the service,
2. Implement a quality-of-service policy,
3. Provide measurements that can help selecting resources under metric service level agreements,
4. Decide for a service agreement,
5. Preselect a number of resources that will likely fulfill the agreement,
6. Execute the service,
7. Evaluate the policies by measuring a successful response,
8. Adapt the strategy if it was not successful, to select new resources (i.e, return to Step 5).

In this paper we will focus on Step 8 of this framework. Other aspects are addressed in [2].

Our paper is structured as follows. In Sections 2, 4, and 5, we define the terms trust and reputation and provide an overview of the existing reputation systems for the Grids and their limitations. In Section 3, we present the general requirements of Grid reputation framework and service. In Section 6, 7, and 8, we propose a new algorithm for managing reputation in Grid-based systems and discuss its underlying architecture. After we provide an overview of other related work we summarize future work and conclude our work.

2. Trust and Reputation. In this section we define the basic terminology used throughout the rest of the paper.

2.1. Definition: Entity. For simplicity, we refer to a resource, agent, service, organization, or user as an *entity*. This definition allows us to specify the term “trust in the most general way while applying it to the Grid approach.

2.2. Definition: Entity Trust. As pointed out by many researchers, trust is an ambiguous concept that defies exact definition. Based on economic models [11], however, we can define trust as a commodity for reducing risk in unknown situations. Hence, trust has an important role in enabling interactions in an unfamiliar environment while weighing the risks associated with actions performed in that environment. The protection of

trust through economic incentives is an important factor to allow trust to become a stable commodity. For our proposed framework, trust is the underlying principle that we determined through local or global interactions among entities and their decisions based on it.

2.3. Definition: Virtual Trust. So far we have not discussed the flow relationships between trustors and trustees. If a trust value in a community is assigned to an entity (the trustor) its trust value can be reused by a new trustee who joins the community and adheres in principle to the same values as the community members. In this case we use the term *community trust*, or *virtual trust*.

2.4. Definition: Entity Reputation. Reputation refers to the value we attribute to a specific entity in the Grid, based on the trust exhibited by it in the past. It reflects the perception that one has of another's intentions and norms. Entity reputation provides a way of assigning quality or value to an entity. Reputation is usually associated with a time factor; it is often gained over time, based on qualities attributed to it by evaluations of other entities. In many reputation models, reputation decreases quickly based on adverse behavior.

2.5. Definition: Entity Reputation Service. An entity reputation service is defined as a secure information service responsible for maintaining a dynamic and adaptive trust and reputation metric within a community. Entities in the Grid continuously interact with the reputation service to create a community rating mechanism that cooperatively assists their future decisions based on the overall community experiences.

3. Trust Models. To define a trust model, we need to establish trust requirements, assign trust ratings, and define trust mediation frameworks and algorithms. Because of the diversity of the Grid and its communities, we cannot define a single trust model suitable for every case. Instead, we need to revisit the requirements and the circumstances in which such a trust model brings added value to the Grid infrastructure. Some of the most common ingredients used to design trust models for Grids are neighborhoods, communities, virtual organizations, contracts, branding, and ownership.

3.1. Neighborhoods and Communities as Trust Models. One of the most common trust models is based on the definition of neighborhoods and communities. Here a group of entities form a relationship network that can be used to query about the trust the members have for another entity to be accessed or used. Neighborhoods are typically small peer-to-peer groups where each member typically knows the others. In contrast to this model, communities contain many more members, and it may no longer be possible for a member of the community to know the others. In both groups, however, trust and reputation are established through standards and common views governed by the communities and neighborhoods. Ratings are established through interpersonal communication or through publication on a community-wide scale. A good example of a neighborhood trust model is the close interaction among computational scientists to interpret the outcome of a particular scientific experiment. A good example of a community trust model is the collection and publication of opinions about a particular topic. In some cases trusted neighborhoods are established to provide the community with trust ratings. An example is an editorial board for the publication of articles in a scientific journal. The scientific community pays more attention to an article reviewed by its peers than to an article published on an unmoderated Web page.

3.2. Geography and Political Boundaries as Trust Model. A simple way to establish neighborhoods and communities is to consider geographical distance or political boundaries. Being a citizen of a foreign country will in most cases require special clearance to participate in entities controlled by a government or university as is often the case for supercomputing centers. Geographical constraints may be needed in order to restrict adaptive trust algorithms to a number of entities in close vicinity. This is often the case for certificate authorities that have branches operating in geographical distributed locations to verify the physical existence of a person. Hierarchical Grids such as the TeraGrid or the Physics Data Grid function in such a fashion. Although considered a virtual organization, membership into this organization sponsored by the community is determined by local trust authorities.

3.3. Contracts as Trust Model. A contract is a binding agreement between two or more persons or parties. Contracts are currently under much discussion as part of service level agreements in QoS-based frameworks such as Web services and Grid services. Here a contract between entities is formed and agreements are cast to fulfill a particular service. This concept is based on the trust that the agreement will be fulfilled. If an unrepeatable entity is present, however, the model will not function, and adaptations need to be made to enforce the agreement (e.g., through litigation or punishment). One of the earliest such models used in Grid

computing was experimented with by the Java CoG project in 1997 in a high-throughput structural biology project. Resources were put together in a pool and if a resource failed to report or the average time taken by other resources to respond was above a threshold, that resource was marked as unfavorable and was chosen only if no other resources were available. In other words, the resource obtained a certain reputation based on its contractual fulfillment.

3.4. Ownership as Trust Model. Highperformance computing has traditionally focused on ownership models. Such models are an extension of the community model in which, however, the ownership of an explicit entity forms a community. In the 80s and 90s these models were driven by supercomputer centers that offered their users exclusive use of supercomputers through batch queuing systems. Today, in Grid, the ownership model is the most common one. We believe that in future, however, we will see a shift toward virtual ownerships (as already promoted by the concept of virtual organizations). Not only will we see virtual organizations but we will also see soon virtual memberships to these organizations.

To apply the concept of ownership to community Grids [21], one must revisit the role of virtual organizations, institutions, and members creating them. Since shared resources in a virtual organization are contributed by various institutions, an elaborate reputation service is needed, that deals with the fact that resources can be part of multiple domains and VOs. The different cases are depicted in Figure 3.1. We use the following nomenclature: ${}^n E_i$ defines an entity with the label i that is shared by n organizations. In case we do know a percentage of share, we augment it appropriately ${}^{p_1 \dots p_n} E_i$ where p_k defines the percentage of ownership of organization k . Considering this nomenclature, we can define use of entities based on the reputation entities obtain. We note that entities within organizations can evaluate each other. To make the system work, however, we need to define a value-based system across the organizations or maintain reputation for different communities and virtual organizations.

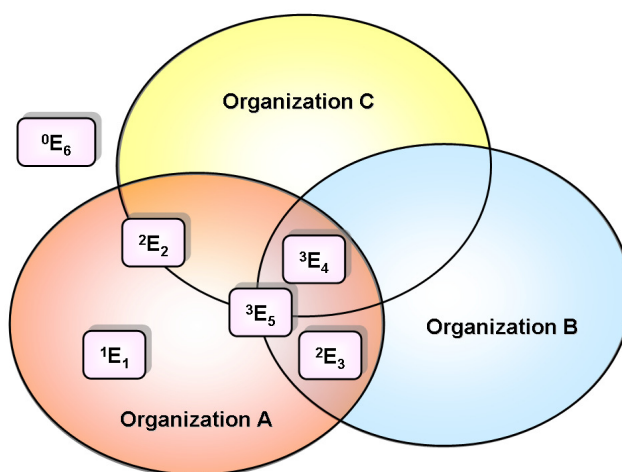


FIG. 3.1. Institutions contribute in various ways their resources and services to possibly various virtual organizations.

3.5. Use as Trust Model. One of the simplest trust models is based on the number of uses. The concept is the following: the more the entity is used, the higher the trust in this resource. Common sense suggests that when so many perceive this entity as desirable, it must be so. Use statistics have long been popular in the computer industry, although these often give a first impression of which entities should be considered, one must make sure that the concept of popularity is independent of other attributes such as security or even content. One need only consider popular but insecure operating systems on Web pages with dubious content appreciated by a large number of Internet users that have achieved more popularity than true content driven pages.

3.6. Branding as Trust Model. One other important concept in industry that is related to reputation is *branding*. Here the reputation of continuously high recommended entities that belong to a particular class or organization may create the desire by other customers to use the same well known brand. Branding is usually in business a good concept as outliers of poor accidental events effecting the reputation negatively are damped.

In computer science the concept of branding is also often used in regards to organizations and products derived from these organizations.

3.7. Time as Trust Model. Time is an essential variable as part of each trust model. Trust and reputation models have sometimes a wide variety of potential conflicting time assumptions. We have branding that clearly augments an entity with a reputation that is less time sensitive than establishing short term contracts between entities that only deal with one time interactions. A similar concept to branding is seniority with time in which the assumed entity becomes a seniority value that is based on experience gained through interaction with the community. Statements such as I have done it this way for years, it must therefore be working for the upcoming years are common.

3.8. Economy as Trust Model. In order to establish a better reevaluation methodology, trust models can be augmented through economic models. For example, contracts can be signed under exchange of real or virtual money, use can be rewarded through a coupon system, and auctions or markets can be put in place to bid for the most trusted and capable resources. This approach naturally can succeed only if a common, controllable commodity such as (virtual) money is used. Business and economic research in these areas is plentiful; indeed the term *virtualization* in business models long before the Grid community used these terms [17].

3.9. Reputation as Trust Model. As indicated earlier, reputation can be used as a major enhancement to each of the models introduced. Since reputation defines a metric, we should be able to use this metric to select entities for closer consideration as part of a neighborhood, community, or virtual organization and help support models employing economic goals, usage, and to establish contracts. This is of especial importance because the time it takes to query all available entities for the best possible fit may be too large. Hence we need to group a class of properties of interest to a particular community and preselect from the many thousands or millions those that give the highest likelihood of success.

4. Application of Reputation Related Trust Models. Trust models and use of reputation frameworks has been considered in a wide variety of systems. The most visible frameworks have been used to enhance business and information services available for a large community through the Internet.

4.1. Review Trust Model. One popular use to establish reputation is to design information portals, similar to C|net [8], which maintains ratings for products based on the ratings of an editor. Integrating feedback from the community provides an additional value in order to reevaluate the judgment of the editor against input from a larger community. Although, the community feedback is not integrated into the editors rating it is still available for review. Hence, the consumer must review both pieces of information to obtain an accurate picture. Detailed textual reviews are also provided to provide the consumer with a semantic explanation on the reason for the given grade by another consumer. The advantages of integrating a community are that the bias of an editor may be minimized. The disadvantage is that invalid responses not corresponding to the editors standard could result in an incorrect evaluation.

4.2. Buyers and Sellers Reputation Trust Model. The online auction system eBay [9] is an important example of successful reputation management. In eBay's reputation system, buyers and sellers can rate each other after each transaction. The feedback system is based on a simple point system, that assigns a positive point for a positive feedback, No points for neutral feedback, and a negative point for a negative feedback. The reputation is the summation of all feedbacks for a buyer or seller over the last six month. Additionally, the feedback is classified in a detailed view to be groups in time periods of the past 7 days, the past month, and the past six month. E-bay points out the a high feedback rating not necessarily means a good reputation. It "is a good sign, but a consumer "should always check a member's feedback profile for any negative remarks. It's best not to judge users only on their feedback ratings.

4.3. Information Ranking. The search engine Google [6, 15] provides a reputation and trust model based on a method called PageRank that uses the links between pages as input. Here a link from other pages to the page in question is interpreted as a positive sign and indicates that the page has some importance. The model is based on the concept that the more links can be found the more important the page is. Additionally, it weighs the pages based on the importance of the voting page.

5. Basis of GridEigenTrust. Before discussing our Grid reputation management framework and the GridEigenTrust algorithm, we provide a short overview of current research efforts that form the basis of our

work. The GridEigenTrust algorithm is inherently based on the peer-to-peer (P2P) EigenTrust algorithm [16] and the use of reputation to define evolving and managed trust in Grids through the introduction of global trust [1]. The GridEigenTrust algorithm combines these algorithms making it conducive for a large Grid environment by increasing its scalability.

5.1. EigenTrust Algorithm for P2P Networks. A reputation management algorithm for P2P networks, called EigenTrust, is introduced in [16]. We summarize the main principle but use within this section the term entity instead of peer in order to provide a uniform nomenclature. Every entity E_i rates other entities based on the quality of service they provide. Therefore, every entity E_j with whom E_i had business will be rated with a grade g_{ij} ($i \xrightarrow{g_{ij}} j$) and is normalized as described in [16]. Hence, for each entity E_j , the normalized local trust value c_{ij} is defined as follows:

$$c_{ij} = \frac{\max(g_{ij}, 0)}{\sum_j \max(g_{ij}, 0)} \quad (5.1)$$

The normalized local trust values throughout the P2P domain needs to be aggregated. This procedure can be done by means of a transitive trust mechanism: entity E_i asks his friends for their opinions about other entities:

$$t_{ij} = \sum_k c_{ik} c_{kj} \quad (5.2)$$

where t_{ij} represents the trust that entity E_i puts in entity E_j based on the opinion of his k friends. The coefficients are assembled into a matrix, $C = [c_{ik}]$. Hence, equation (5.2) can be written in matrix notation as shown in equation (5.3):

$$\vec{t}_i = C^T \vec{c}_i \quad (5.3)$$

The process of obtaining the trust values of friends is repeated to obtain the transitive closure of the matrix. After n iterations, where n is the rank of the matrix, the transitive trust is obtained. For large n , \vec{t}_i converges rapidly as shown in [12], to the same value \vec{t} . Hence, \vec{t} shows how much trust the system as a whole has for every entity E_i .

5.2. Managing Reputation in Grid Networks. In [1, 14] several aspects of trust values are considered as part of a *global* reputation model. In this model it is assumed that the trust values decay with time. It is also assumed that the trust model should stimulate organizations to sanction entities who are not behaving consistently in the Grid environment and who break trust relations. Finally, it is assumed that trust relationships are based on a weighted combination of a *direct* relationship between domains and the *global* reputation of the domains. The model is also based on contexts that, in Grids, can be numerous, varying from executing a specific job, to storing information, downloading data, and using the network. To reflect more accurately the terminology of the Grid, we replace the term domain with organization. We believe that the domain is not an appropriate division for trust within Grids.

Our goal is to define a formula for the trust relationship function Γ , based on the parameters time, context, and the organizations involved.

- Let O_i and O_j denote two organizations.
- Let $\Gamma(O_i, O_j, t, c)$ denote a trust relationship based on a specific context c at a given time t of O_i toward O_j .

Next we define Γ with the help of the following functions:

- Let $\Theta(O_i, O_j, t, c)$ denote a direct relationship for the context c at time t of O_i towards O_j , which is the relationship between neighboring organizations that have direct relationships between entities in both.
- Let $\Omega(O_j, t, c)$ denote the global reputation of O_j for the context c at time t .
- Let $DTT(O_i, O_j, c)$ denote a direct trust table entry of O_i for O_j for context c . The table records the trust value from the last transaction between O_i and O_j .

- Let $\Upsilon(t - t_{ij}, c)$ denote the decay function for specific context c , where t is current time and t_{ij} is the time of the last update of DTT or the time of the last transaction between O_i and O_j .

In [1, 14], $\Gamma(O_i, O_j, t, c)$ is computed as the weighted sum of direct relationship between domain and global reputation of the domain:

$$\Gamma(O_i, O_j, t, c) = \alpha \cdot \Theta(O_i, O_j, t, c) + \beta \cdot \Omega(O_j, t, c) \quad (5.4)$$

where $\alpha, \beta \geq 0$, $\alpha + \beta = 1$.

The direct relationship is affected by the time elapsed between interdomain contacts, hence

$$\Theta(O_i, O_j, t, c) = DTT(O_i, O_j, c) \cdot \Upsilon(t - t_{ij}, c) \quad (5.5)$$

The global trust for domain O_j is computed as

$$\Omega(O_j, t, c) = \frac{\sum_{k=1}^n R(O_k, O_j) \cdot RTT(O_k, O_j, c) \cdot \Upsilon(t - t_{kj}, c)}{\sum_{k=1}^n (O_k)} \quad (5.6)$$

where $R(O_k, O_j)$ is the recommender's trust level, and RTT is usually equal to DTT. Since reputation is based primarily on what organizations say about another domain, the recommender's trust factor $R(O_k, O_j)$ is introduced to prevent cheating through collusion among a group of domains. Hence, $R(O_k, O_j)$ is a value between 0 and 1 and will have a higher value if O_k and O_j are unknown or have no prior relationship among each other and a lower value if O_k and O_j are allies through, for example, a virtual organization relationship.

6. GridEigenTrust Framework. In this section we introduce more details about our proposed GridEigenTrust framework. We begin by pointing out some of the limitations of the two other approaches discussed in Section 5. Then, we show how one can build a more advanced framework by combining the two approaches, while avoiding their limitations while applied to the Grid.

The eigenvalue approach discussed in 5.1 is explicitly designed for P2P networks. It has not been applied to the underlying architecture of Grids that introduce virtual organizations, providing an obvious classification of resources, users, and their reputation that is needed to establish scalability. The approach discussed in [1] has several limitations. First, as already pointed, the use of the term domain is not appropriate for Grids. Hence we have modified the original formulation as shown in Section 5.2. Second, in case of a large number of organizations, it will be costly to compute the global trust (Equation 5.6) because we will have to consider all relationships to increase accuracy. To improve scalability, one can compute the global trust among a set of neighbors; however, such a computation would represent only trust between neighbors but not a global trust value. Third, the authors suggest in their study limiting the number of contexts on. Specifically, the authors reduced the number of contexts in the study to only three: printing, storage, and computing. In Grid environments, however, we deal with many more contexts. An example is the evaluation of trust and reputation for network characteristics, an essential part of any Grid infrastructure. Fourth, the function Υ , which depends on the duration of the interaction between two organizations, must be chosen carefully. We believe that for contexts such as file transfer, a time decay function may have to be chosen far larger than the longest file transfer to be considered, otherwise the decay function may invalidate the reputation even before the transaction is completed. Hence, it will be necessary to introduce classes of similar context, for example, for file transfers with different numbers of bytes. Another limitation is that in the case of networks the actual speed between resources could vary, making it even more complex to obtain the proper trust values.

We design a new algorithm, called Grid EigenTrust, that overcomes some of the limitations of these two approaches. We apply the EigenTrust algorithm explained in Section 5.1 to address the problems of scalability and multiple contexts; at the same time we introduce a global trust value based on the ability of institutions to maintain a trusted Grid environment and provide the high-performance community with reputation services.

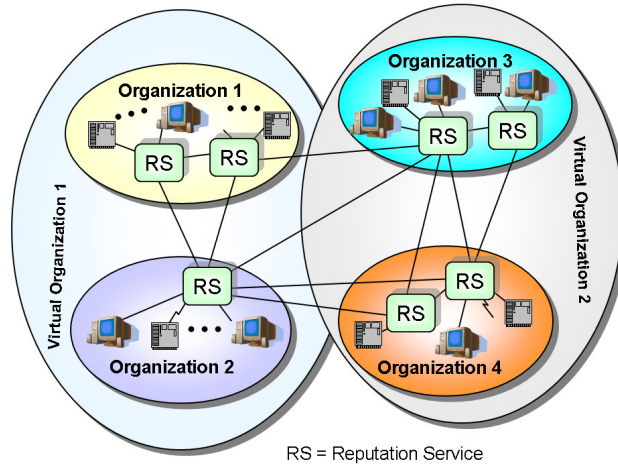


FIG. 6.1. Example of a distribution of reputation management framework based on reputation services in a Grid.

7. GridEigenTrust Algorithm. We address the complexity issue by introducing a set of reputation services arranged in hierarchical graphs. To illustrate this point, we consider the scenario shown in Figure 6.1.

In this scenario, two VOs are depicted containing two organizations each. Each organization has a set of entities. Hence, we have introduced an implicit hierarchy based on entities, institutions, and virtual organizations. We assign a reputation to the entities in the lowest level. Based on the reputation of the entities, the reputation of the organization gets updated. Finally we compute the reputation of a virtual organization by using the reputation values of all the organizations that belong to the virtual organization. Our reputation service can be reused and integrated in each level of the hierarchy.

The number of reputation services needed for a virtual organization or institution may vary based on its implicit size, determined by the entities and the hierarchy they define. Each reputation service is responsible for a subset of entities within the hierarchy. The reputation services compute the reputation in a collaborative, but distributed, fashion. Under the assumption that the interchange of reputation data is secure and can not be compromised, and the time interval that a datum is valid is longer than the smallest update, it may be possible to distribute previous reputation values from entities in the network in order to reduce the network overhead for lookups through a simple caching mechanism. In order to calculate and maintain the reputation, each reputation service uses the GridEigenTrust algorithm described in the next section. To guarantee accuracy, the reputation services must exchange messages with each other in a secure way and the semantics of the reputation service must be secured through a service signature that can be used to clearly identify whether the service has been tampered with.

7.1. Calculating Trust. To describe our GridEigenTrust algorithm, we use the notation used in Section 5.2. To simplify our discussion, we assume each entity is in only one organization (compare Section 3.1).

We establish a trust value for each entity based on various contexts it supports within an organization. We use the term *organization trust* to refer to a trust value for each organization. *Organization trust* differs from other context trust in that it agglomerates several context trust values to a single one. It reflects a general opinion of the reliability of an organization to provide accurate information on what resources this organization supplies. As a result, a reliability trust between organizations can be calculated quickly to obtain the global trust. Although this strategy sounds initially counterintuitive, it is often used in an economic model based on the trust model through branding.

By combining organization trust and the trust level of an entity within an organization (for a specific context c at time t), we derive a reliable trust value for the given entity. We apply the eigenvector mathematical model to compute the global reputation of an organization. Currently, we compute the reputation of a virtual organization as weighted sum of the reputations of all organizations that belong to the virtual organization.

7.1.1. Calculating the Trust of Entities. To describe how an organization maintains trust parameters of its entities, we modify the notation from Section 5.2. Since we are calculating trust values *locally*, (i.e. within

an organization), we omit the first parameter in the function specification Θ , which denotes the entity from which the trust value was obtained.

All entities that use resources or collaborate with users within another organization grade the quality and reliability of the requested entity. The overall grade of the entity is established as the weighted sum of the previous grade (which decays with time) and the new grade. It is also important to consider how much we trust the organization from which the remote entity (i.e., entity that gives the grade) originates its requests.

If $\Theta_p(E_i, t_i, c)$ is the previous cumulative grade established at time t_i for entity E_i within context c , then $g_{ij}(t, c)$ is a new grade given by entity from organization O_j , and $T(O_j)$, then reliability trust level of organization O_j , is the overall new cumulative grade. Then, $\Theta(E_i, t, c)$ can be calculated as

$$\Theta(E_i, t, c) = \frac{\alpha(c) \cdot \Theta_p(E_i, t_i, c) \cdot \Upsilon(t - t_i, c) + \beta(c) \cdot T(O_j) \cdot g_{ij}(t, c)}{\alpha(c) + \beta(c)} \quad (7.1)$$

where $\alpha(c), \beta(c) \geq 0$.

Equation 7.1 is similar to Equation 5.5 from Section 5.2. However, the parameters $\alpha(c)$ and $\beta(c)$ reflect the context importance of the latest grade the entity received.

If an organization just joined the Grid, the initial trust values will be set to a low initial value because the trust must be earned first. However, if the entity for which we assign the trust is sufficiently similar to others in the already existing Grid, an initial value can be obtained from these integrated entities. We chose the lowest trust value and add as penalty a linear correction function.

Let $\Theta_0(E_i, t_0, c)$ denote the initial trust value for an entity E_i within our organization for a context c . Let $\Theta(E_i, t_i, c)$ denote the cumulative reputation value gathered from other entities (defined by equation (7.1)). Then the initial trust of the entity is the weighted sum of these two values:

$$\Gamma(E_i, t, c) = \frac{\gamma(c) \cdot \Theta_0(E_i, t_0, c) + \delta(c) \cdot \Theta(E_i, t_i, c)}{\gamma(c) + \delta(c)} \quad (7.2)$$

where $\gamma(c), \delta(c) \geq 0$.

7.1.2. Calculating the Reliability Trust between Organizations. The reliability trust of organization O_i toward organization O_j reflects *the opinion of organization O_i about the quality and trustworthiness of information organization O_j supplies*. Therefore, besides maintaining individual contexts, we introduce global context (compare Section 5.2). We use a similar notation as in Section 5.2, but we omit the parameter c . If we have a priori knowledge about the initial trust information, we assign this value at initialization time of our algorithm.

Let the initial value of trust be represented as $C(O_j)$. *Reliability* trust should be obtained through the weighted sum of direct experience and global trust value of organization O_j .

Direct experience can be calculated in the same way as in equation 7.1. It is a normalized weighted sum between $C(O_j)$, the cumulative grade from the previous period $\Theta_p(O_i, O_j, t_{ij})$ and the new grade $G(t)$.

Users within organization O_i grade the reputation of a certain entity E_j within organization O_j with grade $\Phi(E_j)$. Also, organization O_j advertises the quality-of-service of this entity with grade $\Delta(E_j)$. Then, organization O_i will grade reliability of information given by organization O_j with grade $G(t)$. For determining grade $G(t)$ we have three cases:

- If $\Phi \in [\Delta - \epsilon, \Delta - \zeta]$, the new grade $G(t)$ is 1.
- If $\Phi > \Delta - \zeta$, the new grade $G(t)$ is bigger than 1.
- If $\Phi < \Delta - \epsilon$, the new grade $G(t)$ is less than 1, depending on how much the Φ differs from Δ

Direct experience that organization O_i has with O_j at some time t , $\Theta(O_i, O_j, t)$ can be calculated in the same way as in equation 7.1. It is a normalized weighted sum between $C(O_j)$, cumulative grade from the previous period $\Theta_p(O_i, O_j, t_{ij})$ and the new grade $G(t)$.

$$\Theta(O_i, O_j, t) = \frac{\alpha \cdot C(O_j) + \beta \cdot \Theta_p(O_i, O_j, t_{ij}) \cdot \Upsilon(t - t_{ij}) + \gamma \cdot G(t)}{\alpha + \beta + \gamma} \quad (7.3)$$

where $\alpha, \beta, \gamma \geq 0$.

Global reliability trust of organization O_j , $\Omega(O_j, t)$ can now be calculated with the EigenTrust algorithm explained in Section 5.1. If we replace c_{ij} with $\Theta(O_i, O_j, t)$ in Section 5.1, we obtain a matrix $C = [\Theta(O_i, O_j, t)]$, and initial vector $\vec{T}_0 = t_0(i)$, $t_0(i) = C(O_i)$. Now we have all the ingredients to apply a power iteration for computing the principal eigenvector of C^T , which represents global reliability trust values for organizations in Grids.

We can summarize the basic steps of the algorithm as follows:

Entity E_i within organization O_1 wants to use entity E_j within organization O_2 in the context c at time t .

- Consider the reliability trust of O_2 computed using the EigenTrust algorithm, $\Omega(O_2, t)$.
- Ask E_i about $\Gamma(E_j, t, c)$, the trust value of organization E_j within organization O_2 .
- In calculating the overall trust value for entity E_j , in formula (5.4) replace $\Omega(E_j, t, c)$ with $\Omega(O_2, t) \cdot \Gamma(E_j, t, c)$
- Compute the overall trust for the entity $\Gamma(E_i, E_j, t, c)$ with formulas (5.4) and (5.5).

After computing the trust values, we can compare them to suggest the resource with the highest reputation. Modifications, such as the introduction of a statistical selection algorithm based on random variables, are possible.

This combined approach has several advantages. First, the algorithm converges rapidly and introduces less overhead than computing global trust values for individual entities within every context. One of the reasons is that the number of values for computation is not too large because we are computing global trust values of organizations through hierarchies, not an overall pool of individual entities. Second, organizations will make an effort to report accurate trust information about their entities because wrong information will be penalized, lowering the global trust of the organization.

8. Reputation Service Architecture. The architecture of an individual reputation service is shown in Figure 8.1. It consists of a collection manager, computation manager, storage and collection manager, and reporter. The collection manager is responsible for evaluating the quality statement describing the requested reputation, and collecting relevant data from the entities such as resources and users. It gives the collected data to the computation manager. The computation manager computes the reputation values of entities based on the context specified and gives the result to the storage manager, which stores the values to maintain a global and historical view. The reporter contacts the storage manager to report the reputation values whenever queried by some entity in the Grid.

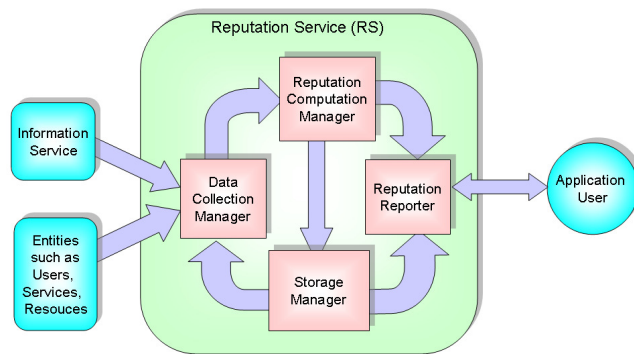


FIG. 8.1. Architecture of a reputation service.

Hence, when an application submits a request for a service cast in a qualitative statement to the reputation service, the reputation service evaluates the statement and computes the reputation for all the entities providing the required service using the heuristics explained in Section 7.1. It contacts other reputation services if required and returns the information regarding the services and their reputation back to the requester. The requester can decide to select the service by looking at the reputation values. This procedure can be easily modified for enabling and enhancing automating resource selection decisions in the Grid.

9. Conclusion and Future Work. In this paper, we have described a framework for calculating reputation in Grid-based system. The paper was mostly focused on issues that have to be addressed while working toward Grid services that integrate reputation concepts into their functionality. We have identified several of these issues. Second we have experimented with an architecture and algorithm to gain experience with this new area of research for the Grid community. We have identified a framework and algorithm, that is a combination of other research efforts. The underlying algorithm is based on introducing a global trust value that is updated with an eigenvalue based trust calculation algorithm. At present we are enhancing and evaluating our framework by introducing a variety of reputation measurements that are controlled through adaptive parameters.

Acknowledgment. This work was supported by the Mathematical, Information, and Computational Science Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-Eng-38. DARPA, DOE, and NSF support Globus Alliance research and development. The Java CoG Kit Project is supported by DOE SciDAC and NSF Alliance. We thank Dr. Paul E. Plassman and Kaizar Amin for detailed and insightful comments on the paper. The algorithm described in this paper was first presented by Gregor von Laszewski in the Workshop on Adaptive Grid Middleware, in New Orleans, LA, September 28, 2003. A draft publication of part of the material that was presented is authored by B. K. Alunkal I. Veljkovic, G. von Laszewski, and K. Amin. This draft is available from <http://www.mcs.anl.gov/~gregor>

REFERENCES

- [1] *Evolving and Managing Trust in Grid Computing Systems*, Hotel Fort Garry, Winnipeg, Manitoba, Canada, May 12-15 2002, IEEE Computer Society Press. Available from World Wide Web: http://www.cs.mcgill.ca/~anrl/PUBS/ccece2002_farag.pdf.
- [2] R. AL-ALI, K. AMIN, G. VON LASZEWSKI, O. RANA, AND D. WALKER, *An OGSA-based Quality of Service Framework*, in GCC2003, Shanghai, 2003. Available from World Wide Web: <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--qos.pdf>.
- [3] *amazon*. Web page. Available from World Wide Web: <http://www.amazon.com>.
- [4] K. AMIN, M. HATEGAN, G. VON LASZEWSKI, A. ROSSI, S. HAMPTON, AND N. J. ZALUZEC, *GridAnt: A Client-Controllable Grid Workflow System*, in 37th Hawai'i International Conference on System Science, Island of Hawaii, Big Island, 5-8 Jan. 2004. Available from World Wide Web: <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gridant-hics.pdf>.
- [5] F. BERMAN, G. C. FOX, AND T. HEY, eds., *Grid Computing: Making The Global Infrastructure a Reality*, Wiley, 2003.
- [6] S. BRIN AND L. PAGE, *The Anatomy of a Large-Scale Hypertextual Web Search Engine*, Computer Networks and ISDN Systems, 30 (1998), pp. 107-117. Available from World Wide Web: <http://www-db.stanford.edu/~backrub/google.html>.
- [7] *Internetworking technology handbook, quality of service*. Web Page, visited Dec. 2004. Available from World Wide Web: http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm.
- [8] *Cnet*. Web Page. Available from World Wide Web: <http://www.cnet.com>.
- [9] *ebay*. Web page. Available from World Wide Web: <http://www.ebay.com>.
- [10] I. FOSTER AND C. KESSELMAN, eds., *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, Dec. 2003.
- [11] D. GAMBETTA, ed., *Trust*, Blackwell, 1990, ch. Chapter 4: Trust as a Commodity, pp. 49-72.
- [12] T. H. HAVELIWALA AND S. D. KAMVAR, *The Second Eigenvalue of the Google Matrix*. Web page. Available from World Wide Web: <http://www.stanford.edu/~sdkamvar/papers/secondeigenvalue.pdf>.
- [13] W. HEISENBERG, *Über quantentheoretische umdeutung kinematischer und mechanischer beziehungen*, Zeitschrift für Physik, 33 (1925), pp. 879-893.
- [14] THE INTERNATIONAL ASSOCIATION FOR COMPUTERS AND COMMUNICATIONS, *Integrating Trust into Grid Resource Management Systems*, Vancouver, B.C., Canada, Aug. 18-21 2002, IEEE Computer Society Press. Available from World Wide Web: http://www.cs.umanitoba.ca/~anrl/PUBS/icpp2002_farag.pdf.
- [15] S. D. KAMVAR, T. H. HAVELIWALA, AND G. H. GOLUB, *Adaptive Methods for the Computation of Page Rank*. Web page. Available from World Wide Web: <http://www.stanford.edu/~sdkamvar/papers/adaptive.pdf>.
- [16] S. D. KAMVAR, M. T. SCHLOSSER, AND H. GARCIA-MOLINA, *The eigentrust algorithm for reputation management in p2p networks*, in Twelfth International World Wide Web Conference, 2003, Budapest, Hungary, May 20-24 2003, ACM Press. Available from World Wide Web: citeseer.nj.nec.com/article/kamvar03eigentrust.html.
- [17] A. MOWSHOWITZ, *Virtual organization: a vision of management in the information age*, The Information Society, 10 (1994), pp. 267-288.
- [18] G. VON LASZEWSKI AND K. AMIN, *Middleware for Communications*, in Grid Middleware, Wiley, 2004, pp. 109-130. Available from World Wide Web: <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--grid-middleware.pdf>.
- [19] G. VON LASZEWSKI, S. FITZGERALD, I. FOSTER, C. KESSELMAN, W. SMITH, AND S. TUECKE, *A Directory Service for Configuring High-Performance Distributed Computations*, in Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing, 5-8 Aug. 1997, pp. 365-375. Available from World Wide Web: <http://www.mcs.anl.gov/~gregor/papers/fitzgerald--hpd97.pdf>.
- [20] G. VON LASZEWSKI, J. GAWOR, C. J. PEÑA, AND I. FOSTER, *InfoGram: A Peer-to-Peer Information and Job Submission Service*, in Proceedings of the 11th Symposium on High Performance Distributed Computing, Edinbrough, U.K.,

- 24-26 July 2002, pp. 333–342. Available from World Wide Web:
<http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--infogram.ps>.
- [21] G. VON LASZEWSKI AND P. WAGSTROM, *Gestalt of the Grid*, in Tools and Environments for Parallel and Distributed Computing, Series on Parallel and Distributed Computing, Wiley, 2004, pp. 149–187. Available from World Wide Web:
<http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--gestalt.pdf>.
- [22] G. VON LASZEWSKI, M. WESTBROOK, I. FOSTER, E. WESTBROOK, AND C. BARNES, *Using Computational Grid Capabilities to Enhance the Ability of an X-Ray Source for Structural Biology*, Cluster Computing, 3 (2000), pp. 187–199. Available from World Wide Web: <http://www.mcs.anl.gov/~gregor/papers/vonLaszewski--dtrek.pdf>.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 8, 2003.

Accepted: September 1, 2003.



NON-DEDICATED DISTRIBUTED ENVIRONMENT: A SOLUTION FOR SAFE AND CONTINUOUS EXPLOITATION OF IDLE CYCLES

R. C. NOVAES* , P. ROISENBERG* , R. SCHEER* , C. NORTHFLEET* , J. H. JORNADA* , AND W. CIRNE†

Abstract. The Non-Dedicated Distributed Environment (NDDE) aims to muster the idle processing power of interactive computers (workstations or PCs) into a virtual resource for parallel applications and grid computing. NDDE is novel in the sense that it allows for safe and continuous use of idle cycles. Differently from existing solutions, NDDE applications run inside a virtual machine rather than on the user environment. Besides safe and continuous cycle exploitation, this approach enables NDDE applications to run on an operating system other than that used interactively. Our preliminary results suggest that NDDE can in fact harvests most of the idle cycles and has almost no impact on the interactive user.

Key words. Grid Computing, Virtual Machines, Idle Cycles.

1. Introduction. Modern desktop computers and workstations have powerful computational capabilities that are used primarily to provide short response times to the user's daily activities like word processing, spreadsheet calculations or web page rendering. Most of the time, however, this processing power is idle, waiting for occasional user inputs or requests. During this unused periods that can range from fractions of a second (e.g. between user keystrokes) to hours, the operating system normally executes an "idle" process, which is a dummy process with the lowest priority on the system, so it runs only when there is no other process or service needing to be executed. The processing capacity used to run this idle process is in fact being wasted.

The Non-Dedicated Distributed Environment (NDDE) aims to potentially use all of this fragmented idle time from most or all the machines connected to a network. This will create a very low cost virtual resource with only minimal interference on the normal operation of the interactive users. Such a virtual resource can be directly used to run parallel application or can be a component of a large computational grid.

Of course, this is not a new idea. Systems like Condor [1] and SETI@home [2] are classic examples of successful exploitation of idle cycles to do useful computation. NDDE differs from these in the sense that it allows for safe and continuous use of idle cycles. It is safe because it is much harder for a malicious guest application to tamper with user data and environment. It is continuous because it avoids the "interactive versus idle" resources dichotomy. That is, NDDE enables both environments to run concurrently so the workstation does not need to be totally idling to make its resources available. It can exploit idle resources in a much finer grain.

Safe and continuous idle exploitation is possible because NDDE applications run inside a virtual machine rather than on the user environment. Note that, additionally, this approach enables NDDE applications to run on operating system other than that used interactively.

We have conducted some initial experiments to (i) gauge how much of the idle cycles NDDE can in fact deliver for a parallel or grid application, and (ii) measure its impact on the interactive users. In a nutshell, NDDE can in fact harvests most of the idle cycles and has almost no impact on the interactive user. However, it displays a noticeable overhead for I/O intensive applications.

The remainder of this paper is organized as follows. The next section surveys the state of the art in exploring idle cycles. Then, we introduce NDDE, presenting its features and architecture. Finally, we give a performance overview of this environment and conclude with an outlook on future work.

2. Exploiting Idle Cycles. The use of many resources to tackle a single problem dates back (at least) to the 1970's. The conventional approach since then has been to use dedicated platforms for running parallel applications. These platforms are generally assembled as parallel supercomputers (such as IBM SP2 and Cray T3E) or dedicated Beowulf clusters [3].

On the other hand, there are also applications that can use non-dedicated resources, running opportunistically when resources are idle. Since non-dedicated resources are much cheaper than dedicated resources, much effort has been spent to ease using such idle resources. Therefore, we have seen in the 1980's the introduction of systems such as Condor, which enabled parallel applications to effectively benefit from cycles that

*Hewlett-Packard Brazil {reynaldo.novaes, paulo.roisenberg, roque.scheer, caio.northfleet, joao.jornada}@hp.com.

†Federal University of Campina Grande. walfredo@dsc.ufcg.edu.br.

would otherwise be wasted. More recently, SETI@home showed that this approach could scale up to planetary proportions.

However, in traditional idle harvesting systems as can be seen in Condor and SETI@home, the guest parallel application runs in the user environment (i.e., as a process in the user operating system). This creates a security concern. Since the parallel application runs as a normal process inside the user's environment, it may be able to exploit some security breaches and cause damage. There are two possible solutions for this problem. The first is to execute the guest application in an emulated platform, like Java. The second is to reboot the machine and run a completely independent operating system from where the guest application has no access to the user environment.

Systems like HP's I-Cluster [4] and vCluster [5] implement a solution based on reboot motivated by security concerns. These systems, upon detecting that there is no user activity, reboot the machine, entering in a different, separated operating system, in which the guest application runs. This approach requires a separated partition to hold the parallel environment and it addresses the security concerns providing a separated operating system and file system, preserving user data. As an extra advantage, the parallel application can run on an operating system different from the one that serves the interactive user. For example, in I-Cluster and vCluster, Windows caters for interactive users, while parallel applications run on Linux.

One drawback of this approach is that it requires a reboot to switch between the two operating systems and this operation has an impact on the interactive user. This is because switching between operating systems is not instantaneous. It takes tens of seconds, in the best case. In order to minimize such an impact, I-Cluster and vCluster keep track of the usage of the machine to try to predict when the interactive user will need it again. This prediction is used to avoid rebooting the machine into cluster mode when the user is expected to go in activity soon, as well as to reboot back into interactive mode in anticipation of the user's need. Of course, any user activity also prompts the switch back to the interactive operating system.

Other systems, which run guest applications concurrently with local user applications like SETI@home, use a different approach for harvesting idle cycles. They monitor user activity using operating system features, like a screensaver.

However, no matter which approach the system uses, it will always try to minimize the impact in the interactive user. Therefore, the prediction of the user idleness is crucial for switching and concurrent approaches. In a perfect world, the user should not notice the exploitation of idle cycles. An issue that complicates matters is that sometimes the user is not interacting with the machine but she is waiting for a task to be completed, like a download. This means that idleness detection mechanism must monitor many of the system's parameters to correctly detect user activity.

Unfortunately, the above approaches impose a limit on how many idle cycles one can harvest. First, for idleness prediction based systems, idle cycles will be really wasted because in order to cause minimal impact on the user, the system has to be somewhat conservative, keeping the system in interactive mode. Secondly, for screensaver-based systems, the user might be interacting with the computer, but using only a small fraction of its processing power (e.g., when the user is typing text) but the system will be seen as active. In both cases idle cycles will be being wasted.

In short, using the screensaver or rebooting the machine to safely exploit idle cycles seems to be effective when there are big chunks of idle time. Such schemes are not effective at harvesting fragmented idle time.

The NDDE addresses these problems. It allows for the safe exploitation of idle cycles, just as I-Cluster and vCluster, but is also able to harvest fragmented idle time, unlike I-Cluster and vCluster. Another feature that distinguishes our approach from the implementations listed above is that, being based on a virtual machine, it can provide a more homogenous execution environment.

3. NDDE. The NDDE is part of a group of projects hosted by HP Brazil that aim to provide simple solutions for exploiting unused computational resources for grid or cluster usage. The target research subjects are non-dedicated computers in corporations and educational institutions. This research includes the development of environment switching processes using reboot or in concurrent mode, like the solution presented here.

NDDE improves upon original I-Cluster and vCluster projects. It presents a different approach to explore idle time, based on the premise that there are unused cycles even when the user is interacting with the computer. The NDDE implements a virtual machine inside the user's system, running a separated operating system that has its own address space and file system. The parallel applications run concurrently with the user's applications. Although the core of this idea (grid computing using virtual machines) is not new, as we can see in paper of

Figueiredo et al. [6], our contribution is that we propose to increase the availability of a non-dedicated machine to the grid or cluster using as many idle cycle as possible, with minor impact on the interactive user's activities.

Using this approach the system does not require any special action, like rebooting, to become an active cluster resource. The guest operating system runs in a user-mode virtual machine, which has restricted access to user's system resources, thus parallel applications can be safely executed. In order to isolate the user's environment, the virtual machine can only access data inside its file system that is entirely contained inside a single file on user's machine. The main advantages on this approach are:

1. The guest environment is isolated from the user environment. The applications running on the guest OS have their own address and storage space and the access to system resource is made through a software layer provided by the virtual machine.
2. There is no noticeable switching time between the two different environments (user and parallel).
3. There is no instruction set conversion, only system calls conversion. So the overhead for CPU intensive applications is minimum.
4. The user does not need to be aware about exploitation of idle cycles. The only requirement may be that the user should leave the machine always turned on.
5. It increases the availability of the node to be exploited as a cluster resource. Any idle time, no matter how small it is, can be used to perform cluster tasks.

Fig. 3.1 shows the basic architecture model for the solution. The virtual machine acts like a native application and runs concurrently with other applications on the user's machine. The virtual machine can be implemented using open source tools like Plex86 [7] or commercial products like VMware [8]. Another option is to use User-Mode Linux for Windows (Umlwin32) [9], but it lacks the security offered by the virtual machine implementations. In all cases, the user machine's resources are shared between the native applications and the virtual machine.

The virtual machine runs its own instance of an operating system, called 'guest system', that provides access to the virtual machine's emulated storage space and controls the use of other resources like virtual memory space and network access. All parallel application accesses to system resources are made through the Host System Call converter, which converts the virtual machine system calls to equivalent host operating system calls. CPU intensive applications (the typical application on parallel environments) run near native machine speed since there is no machine instruction emulation. The parallel applications are loaded in the virtual machine address space and feel as if they are on a dedicated machine. Note also that parallel applications compatible with the guest operating system do not need to be changed or recompiled to run on this environment.

To improve the security, we could also restrict or completely eliminate the virtual machine's ability to access the network. A trusted application on the host OS would be responsible for transferring code and data in and out from the guest file system. This solution is very similar to Entropia [10] that offers a "sandboxed" environment for safe task execution.

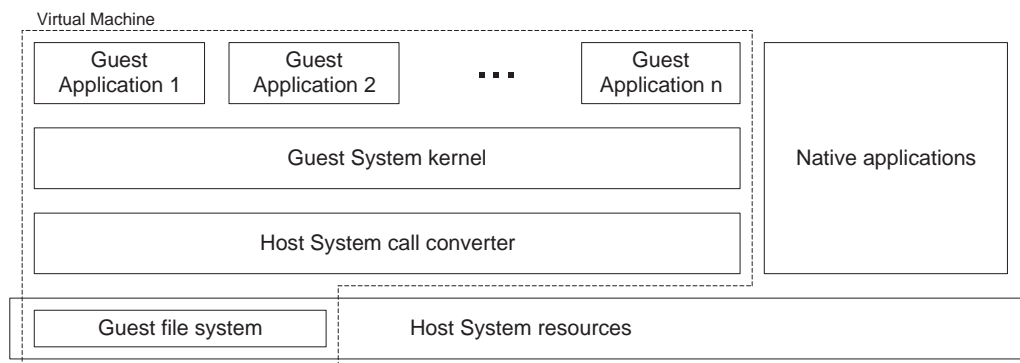


FIG. 3.1. NDDE basic architecture

To not interfere with the regular users of the computers, the virtual machines will be made to run as the process with one of the lowest priorities on the system, having only slightly higher priority than the operating system's own "idle" process. This way the virtual machine will be executed by the operating system only when there is no other process or service able to run, but will be chosen to be run by the operating system

instead of the operating system's own "idle" process. When there are regular applications running, the NDDE environment, composed by the virtual machine and its own applications, will be automatically preempted and maintained by the host operating system in a "ready-to-run" state, so it can continue to run as soon as there is no regular processes or services running.

4. Performance Evaluation. In order to verify the usability of the NDDE two sets of tests were performed. The first one gauges the performance an application can attain via NDDE. The second one measures the impact on the regular interactive usage of the machine.

4.1. Benchmark Environment. The test environment consisted of a pair of HP e-PC 42, a Pentium 4 1.7 GHz machine with 256MB of memory. The host operating system was Microsoft Windows 2000 Professional. The guest operating system was Linux Red Hat 8.0 and used OpenSSI version 0.9.6r3 [11] as the basic parallel processing environment.

At the beginning two implementations for running the guest environment were considered: Umlwin32 and VMware. The VMware was chosen due to the UMLWin32's early development stage. In the tests described here, the guest operating system runs under VMware Workstation version 3.2.0 configured with 128 MB of memory.

4.2. Performance of Parallel Applications. The tests aim to measure the overall performance NDDE makes available for the guest applications. LLCBench, which is a combined set of synthetic benchmarks, was used to make these tests. It is the combination of BLASBench [12], MPBench [13], and CacheBench [14]. MPBench is used to measure the communication performance of MPI [15]. CacheBench has been chosen to determine the virtual machine's memory subsystem performance. Finally, BLASBench is used to measure the performance of a CPU-bound application.

In order to evaluate the performance impact, a baseline test was performed. These tests, referred as 'Native Linux' in the graphs, use machines executing OpenSSI in native mode, without any emulation.

The idea behind these tests was to verify the performance penalties imposed by this approach, that is, an execution environment running concurrently with a completely different operating system. For sure these tests are generic and only basic usability issues are addressed.

The following tests were performed: one group of tests for memory access simulation, shown in Fig. 4.1, Fig. 4.2 and Fig. 4.3, one test for simulating CPU intensive applications, shown in Fig. 4.4 and, finally, a test regarding the network bandwidth, shown in Fig. 4.5.

These graphs show the average results of each test after several runs.

The Fig. 4.1, Fig. 4.2 and Fig. 4.3 show that VMware has some influence on cache operation that is almost constant for all cache size. We speculate that this performance loss is probably due to page fault handling in the virtual machine but further investigation is required to confirm this theory.

The BLAS performance test, shown in Fig. 4.4, also shows that VMware adds little overhead to the guest environment for CPU intensive applications.

The Fig. 4.5 shows that the network operations suffer noticeable losses imposed by the I/O hardware emulation implemented by virtual machine. This happens because the guest application sees a "double OS" on every access to network devices, that needs to be handled first by the guest OS and later again by the host OS. The same situation happens for all file access, as described by Sugerman et al. [16].

These tests show that the virtual machine solution is best suited for CPU-intensive applications but may not be suited for network or I/O intensive tasks.

4.3. Impact on the User. In order to evaluate the impact of NDDE on normal interactive usage, we elected editing a huge file with Microsoft Word 2000 as archetypical representative of a machine's interactive usage. This huge file had 151MB in size, with 2,623,919 words in 14,211 pages. Considering that we were using a machine with 256 MB of memory and VMware was configured to emulate a machine with 128 MB of memory, this file size (151 MB) is expected to cause Microsoft Word Processor to generate some swapping activity.

In the guest operating system, two test applications were developed. One is a CPU consuming application, which executes a continuous loop. The other application is both CPU and memory consuming. This application allocates 100 MB and executes a continuous loop touching every page by changing the contents of a few bytes on each of them to force the pages to be marked as dirty. So, the guest operating system needs to save their contents to the swap file in case it needs to release pages to make room for user applications.

The tests were grouped in four distinct scenarios:

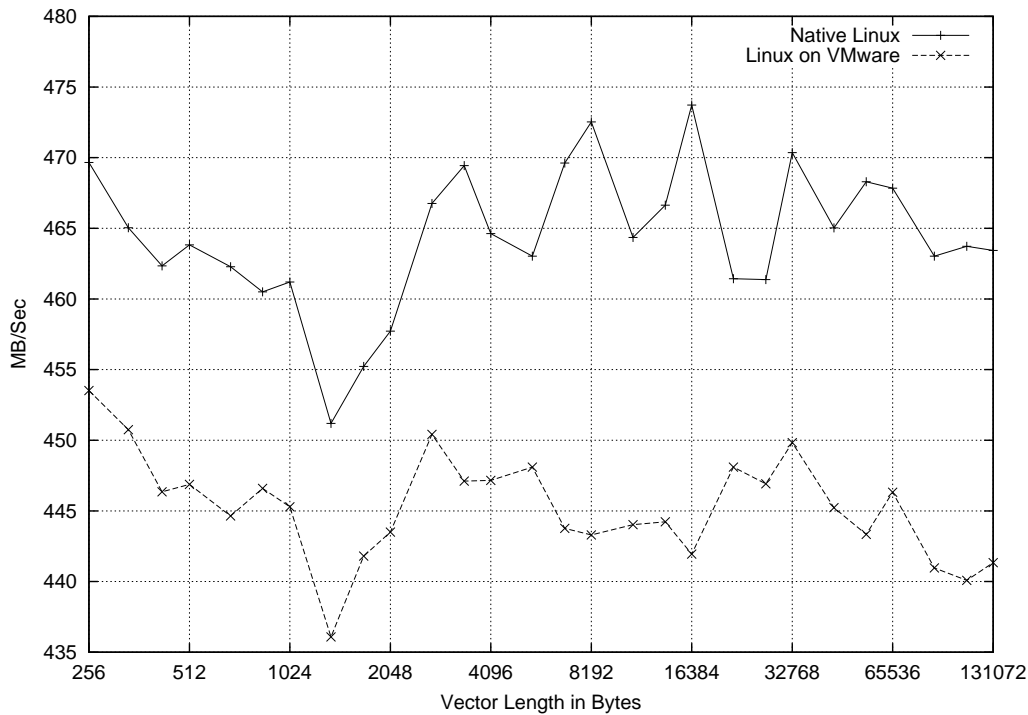


FIG. 4.1. CacheBench Read Performance

1. A baseline machine just with the user application (Microsoft Word 2000 editing the 151 MB file)
2. A machine with the same user application and just the guest operating system executing in VMware (no application executing on it)
3. Same as scenario 2, but executing a CPU bound application in the virtual machine
4. Same as scenario 2, but executing a CPU and memory bound application in the virtual machine

In each scenario, four operations were executed:

1. Starting of the application (Microsoft Word) and huge document load
2. Go to the end of the document
3. Select the "statistics" tab option in document properties
4. Replace a character at the end of document and measuring the time to save it

The completion time for each operation was measured, according to Table 4.1.

TABLE 4.1
Average resulting time (min:sec)

	Baseline	VMware only	CPU bound	CPU + memory bound
Load	0:39	0:41	0:41	0:48
Go to end	5:47	5:56	5:57	6:02
Properties	4:18	4:25	4:26	4:26
Save	3:00	3:10	3:12	3:26

The results shown in Table 4.1 are the mean value of several executions. There is only minor impact on the regular user operation for most scenarios, and even the impact of the concurrency for memory was acceptable in this test. Considering the gain of allowing the machine to act as a cluster node concurrently with normal machine operation, the impact on the regular user side seems to be acceptable.

It is interesting to point out that literature reports cases where the competition for memory introduced by guest application cause serious problems for the interactive use of the machine [17]. This would occur when the interactive applications are sleeping and thus can get swapped out to disk when the guest application needs to allocate more memory. We could not reproduce such a behavior. We conjecture that this is due to the

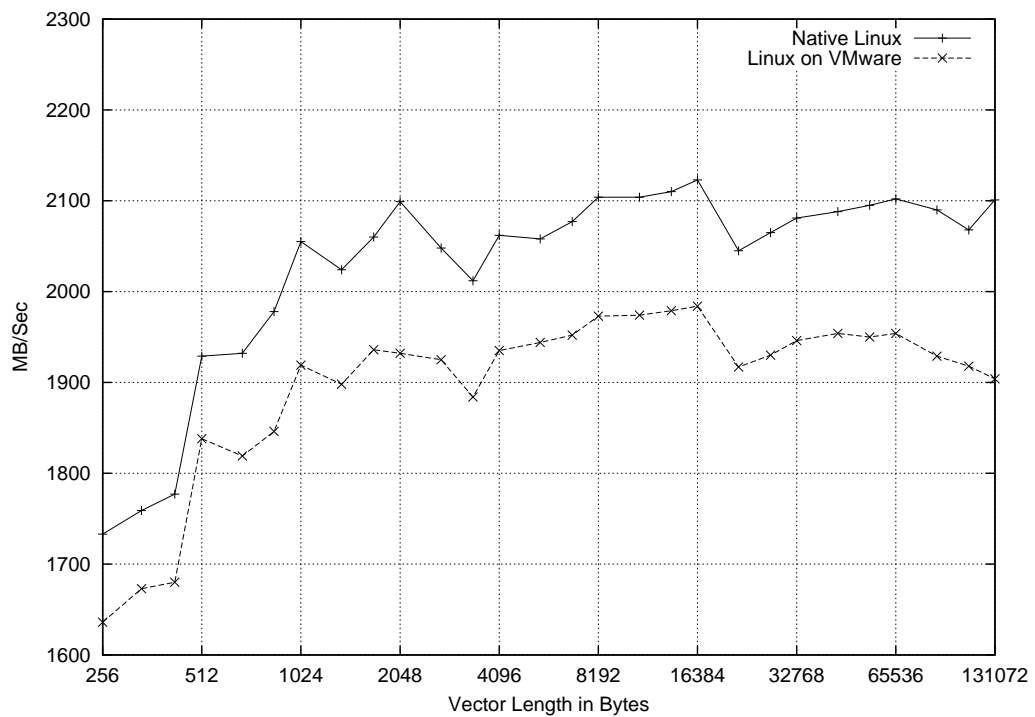


FIG. 4.2. CacheBench Write Performance

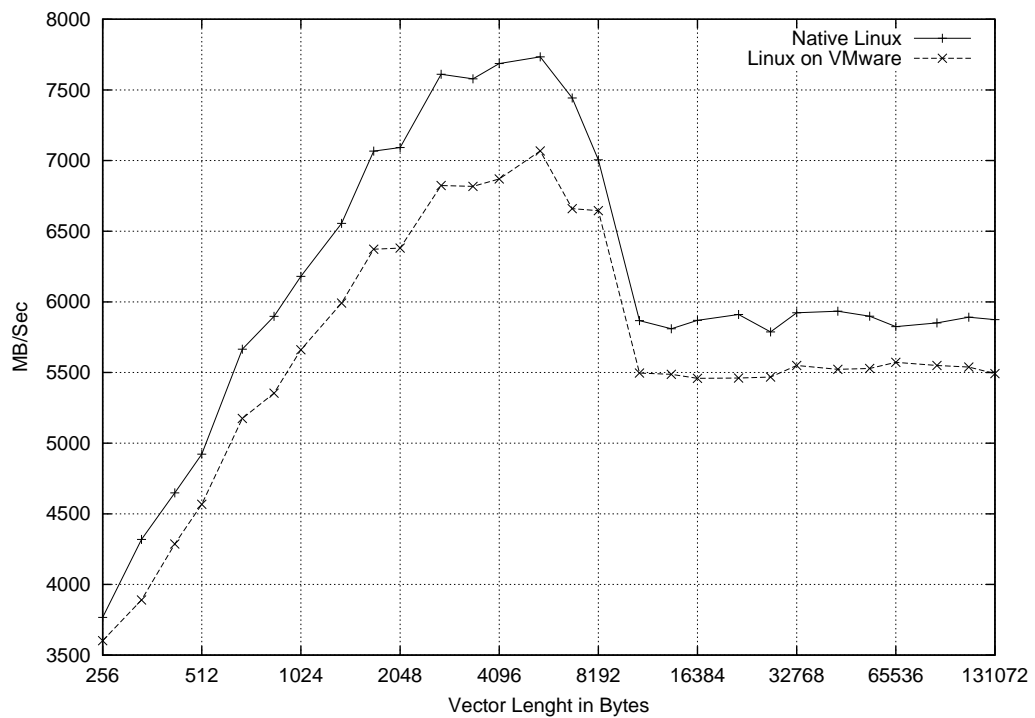
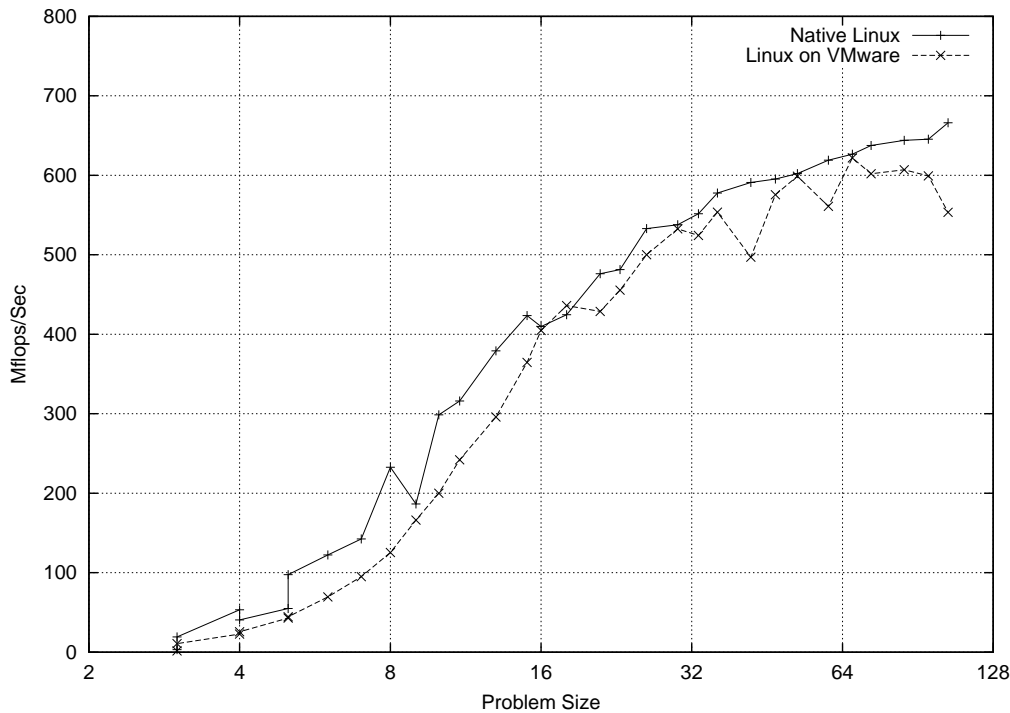


FIG. 4.3. CacheBench Read/Modify/Write Performance

FIG. 4.4. *BLASBench Performance*

fact VMware's memory was limited to 128 MB. In previous experiments reported in the literature, the guest applications had no explicit memory limit.

5. Conclusions. In this article we described NDDE, an alternative way to explore the idle time of interactive computers, turning a set of such computers into a virtual resource for parallel applications and grid computing. NDDE is novel because it allows for safe and continuous use of idle cycles. It is safe in the sense that it is much harder for a malicious guest application to tamper with user data and environment. It is continuous because it can also harvest fragmented idle time. Moreover, since NDDE applications run inside a virtual machine rather than on the user environment, this approach enables NDDE applications to run on an operating system other than that used interactively.

An analysis was carried out to establish the performance of applications that run on NDDE. The results show that NDDE is best indicated when the parallel applications are computationally intensive. Applications that are I/O-intensive may be impacted by the intrinsic limitations of the implementation of virtual machines. The impact on the normal usage of the machine was also measured. The mechanism of using low priority on the virtual machine keeps the impact on the user to a minimum.

The next steps in this work will be going in three directions. First, we intend to evaluate NDDE more thoroughly, refining it where necessary. This includes (i) further investigating the apparent

resilience of NDDE to memory competition with the host applications, (ii) evaluation of the percentage of idle time that is available to be harvested on a typical enterprise or academic network, and (iii) reduce the overhead for parallel applications that are heavily based on internode communications. Some real world data is being collected in order to compare the total of cycles harvested in this solution with a screensaver-based or reboot-based solution.

One relevant result in this work is the performance loss observed in network-bounded applications. This issue motivates us to perform some measurements to determine the application granularity that the guest application should have to make the transference times be acceptable.

All the investigation topics described above will help us to see if this solution is profitable when compared to dedicated and switched environments. Second, it might be worthwhile to combine NDDE's and I-Cluster's approach into a hybrid scheme. For example, most machines are totally idle during the night. We could thus

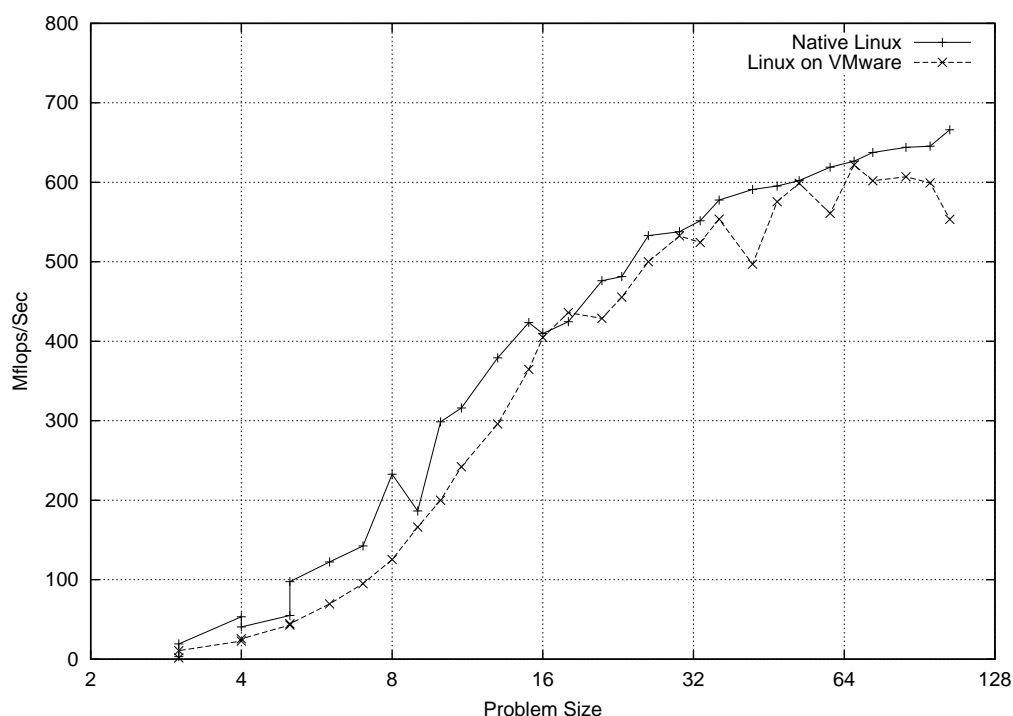


FIG. 4.5. MPBench Bandwidth

think of using I-Cluster during the night and NDDE during the day. Third, we intend to explore NDDE as a sandboxing platform for MyGrid [18], enabling grid applications that cross administrative boundaries. Note that such grid applications raise especially serious security issues, making the use of NDDE technology particularly relevant.

Acknowledgements. The authors wish to express their sincere thanks to the AGridM'03 reviewers for their insightful comments.

REFERENCES

- [1] M. LITZKOW, M. LIVNY AND M. MUTKA, *A Hunter of Idle Workstations*, Proceedings of the 8th International Conference of Distributed Computing Systems, pp. 104–111, June 1988.
- [2] D. ANDERSON, J. COBB AND E. KORPELA, *SETI@home: An Experiment in Public-Resource Computing*, Communication of the ACM, vol. 45, no. 11, pp. 56–61, November 2002.
- [3] D. RIDGE, D. BECKER, P. MERKEY AND T. STERLING, *Beowulf: Harnessing the Power of Parallelism in a Pile-of-PCs*, Proceedings of IEEE Aerospace, 1997.
- [4] B. RICHARD AND P. AUGERAT, *I-Cluster: Intense computing with untapped resources*, MPC'S'02, Ischia, Italy, April 2002.
- [5] C. A. F. DE ROSE, F. BLANCO, N. MAILLARD, K. SAIKOSKI, R. NOVAES AND B. RICHARD, *The virtual cluster: a dynamic environment for exploitation of idle network resources*, Proceedings of 14th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2002), pp. 141–148, Vitória, ES, Brazil, 2002.
- [6] R. J. FIGUEIREDO, P. A. DINDA AND J. A. B. FORTES, *A Case for Grid Computing on Virtual Machines*, Proceedings of International Conference on Distributed Computing Systems (ICDCS), April 2003.
- [7] K. LAWTON, *The new Plex86 x86 Virtual Machine Project*, WWW, August 2003. <http://plex86.sourceforge.net/>.
- [8] VMWARE, *VMware Workstation - Powerful Virtual Machine Software for the Technical Professional*, WWW, April 2003. http://www.vmware.com/pdf/ws_specs.pdf.
- [9] C. KUDIGE, *Umlwin32*, WWW, March 2003. <http://umlwin32.sourceforge.net/>.
- [10] A. A. CHIEN, B. CALDER AND S. ELBERT, *Entropy: Architecture and Performance of an Enterprise Desktop Grid System*, Journal of Parallel Distributed Computing, vol. 63, no. 5, pp. 597–610, May 2003.
- [11] B. J. WALKER, *OpenSSI Linux Cluster Project*, WWW, April 2003. <http://openssi.org/ssi-intro.pdf>.
- [12] NATIONAL SCIENCE FOUNDATION, *BLAS (Basic Linear Algebra Subprograms)*, WWW, March 2003. <http://www.netlib.org/blas/>.
- [13] P. J. MUCCI, K. LONDON AND J. THURMAN, *The MPBench Report*, November 1998. WWW, March 2003. <http://icl.cs.utk.edu/projects/llcbench/mpbench.pdf>.

- [14] P. J. MUCCI, K. LONDON AND J. THURMAN, *The CacheBench Report*, November 1998. WWW, March 2003. <http://icl.cs.utk.edu/projects/llcbench/cachebench.pdf>.
- [15] MESSAGE PASSING INTERFACE FORUM, *MPI: A Message Passing Interface Standard*, May 1994.
- [16] J. SUGERMAN, G. VENKITACHALAM AND B. LIM, *Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor*, Proceedings of the USENIX Annual Technical Conference, June 2001.
- [17] T. E. ANDERSON, D. E. CULLER, D. A. PATTERSON AND THE NOW TEAM, *A Case for Networks of Workstations: NOW*, IEEE Micro, February 1995.
- [18] W. CIRNE, D. PARANHOS, L. COSTA, E. SANTOS-NETO, F. BRASILEIRO, J. SAUVÍ; F. A. BARBOSA DA SILVA, C. OSTHOFF BARROS AND C. SILVEIRA, *Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach*, Proceedings of the ICCP'2003 - International Conference on Parallel Processing, WWW, October 2003. <http://walfredo.dsc.ufpb.br/resume.html#publications>.

Edited by: Wilson Rivera, Jaime Seguel.

Received: July 9, 2003.

Accepted: September 1, 2003.

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 $\text{\LaTeX}2_{\epsilon}$ 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 PDCP 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.