GFD-I.068

Category: Informational

APPS-RG and PGS-RG

David Wallom, University of Oxford
Thilo Kielmann, Vrije Universiteit
Editors

29-03-06

Workshop on Grid Applications: from Early Adopters to Mainstream Users
Held in conjunction with GGF14, (June 26-29 2005, Chicago, USA)

Status of This Memo

This memo provides information to the Grid user community. It does not define any standards or technical recommendations. Distribution is unlimited.

Copyright Notice

1. Abstract

This is the proceedings of the Workshop on Grid Applications that has been organized jointly by the Application Developers and Users Research Group (APPS-RG) and the Production Grid Services Research Group (PGS-RG) of the GGF. It contains the papers that have been accepted for presentation by the programme committee.

Contents

2. Foreword

The objective of the Application Developers and Users Research Group (APPS-RG) is to facilitate the exploitation of grid technology by application developers and users. The Production Grid Services Research Group (PGS-RG) is intended to bring together grid practitioners to document experiences, identify best practice and develop informational documents. In this workshop, we aimed at capturing experience with bridging the gap between early adopters of grids and the more mainstream use of grid technology. Currently, grids are mostly on the early adopter side of the gap, asking for the move to the more mainstream users. For fostering mature production environments, incentives for application users are vital. We were seeking to hear experiences from the following groups:

- Early adopters who would like to become mainstream users,
- Mainstream users who would like to use grids and those who already do,
- Middleware developers and system operators in charge of providing working grid environments to user communities.

Within the remit of the workshop's subject we gave the following guidelines as to the type of contributions that we were expecting:

1. user experience with early-adopter and/or mainstream grid applications
2. management approaches for production-quality grid environments
3. techniques for robust (fault tolerant) grid applications and middleware
4. software tools for automatic testing and signaling of error conditions

The major focus is on hands-on experience with existing grid environments, focusing on bridging the gap between early adoption and production use. This was one of several criteria that were used to assess the suitability of the contributions.

Bearing in mind the short time frame that was available to announce the workshop, the response on submitted papers was very positive. From the submissions, the programme committee selected the 11 papers included in this report. The organizers were very pleased by the interesting presentations with extended versions of the strongest papers appearing in a special issue of the *Journal of Grid Computing* in 2006.

The submissions that were successful fell into two categories;

- Environments, Middleware, and Tools
- Application Experiences

The format of the workshop was to have all talks on a specific area first, then a moderated discussion session afterwards. The summaries of these discussions are in the Section 6.

3. Workshop Organizers

The workshop has been organized by the chair persons of the two GGF research groups which were involved (APPS-RG and PGS-RG):

- Thomas Hinke, NASA Ames, USA, Thomas.H.Hinke@nasa.gov
- Thilo Kielmann, Vrije Universiteit, The Netherlands, kielmann@cs.vu.nl
- Laura McGinnis, Pittsburg Supercomputing Centre, lfm@psc.edu
- Judith Utley, Old Dominion University, jutley@odu.edu
- David Wallom, University of Oxford, david.wallom@ierc.ox.ac.uk

4. Programme Committee

Besides the logistics of organizing a workshop, its success can only be guaranteed by the quality of the submitted papers. To assist with this the following people kindly agreed to review the submitted papers:

Simon Cox	(University of Southampton, UK)
Thomas Hinke	(NASA Ames, USA)
Thilo Kielmann	(Vrije Universiteit, Netherlands)
Pascal Kleijer	(NEC, Japan)
Ignacio Martin Llorente	(Universidad Complutense, Spain)
Laura McGinnis	(PSC, USA)
Andre Merzky	(Vrije Universiteit, Netherlands)
Steven Newhouse	(OMII, UK)
Judith Utley	(Old Dominion University, USA)
David Wallom	(University of Oxford, UK)

5. List of Workshop Papers

- **Production-Quality Grid Environments with UNICORE.** D. Erwin, M. Rambadt, A. Streit, Ph. Wieder, Forschungszentrum Jülich, Germany, Pages 10-17.
- **Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service**. I. Foster, V. Nefedova, L. Liming, R. Ananthakrishnan, R. Madduri, L. Pearlman, O. Mulmo, M. Ahsant, ANL, University of Chicago, ISI, (all USA), KTH Stockholm, Sweden, Pages 18-26.
- **Early Application Experience with the Grid Application Toolkit (GAT)**, S. Le Blond, A. Oprescu, C. Zhang, Vrije Universiteit, The Netherlands, Pages 27-37.
- **Implementation of Fault-Tolerant GridRPC Applications**, Y. Tanimura, T. Ikegami, H. Nakada, Y. Tanaka, S. Sekiguchi, AIST, Japan, Pages 38-49.
- **DMOVER: Parallel Data Migration for Mainstream Users**, N. Stone, B. Gill, J. Kochmar, R. Light, P. Nowoczynski, J.R. Scott, J. Sommerfield, C. Vizino, PSC, USA, Pages 50-59.
- **GeneGrid: Grid Service Based Virtual Bioinformatics Laboratory**, P.V. Jithesh, N. Kelly, S. Wasnik, P. Donachy, T. Harmer, R. Perrot, M. McCurley, M. Townsley, J. Johnston, S. McKee, Belfast e-Science Center, Fusion Antibodies Ltd, Amtec Medical Ltd., UK, Pages 60-68.
- **From Proposal to Production: Lessons Learned Developing the Computational Chemistry Grid Cyberinfrastructure**, R. Dooley, K. Milfeld, C. Guiang, S. Pamidighantam, G. Allen, LSU, NCSA, and TACC, USA, Pages 69-78.
- **Solving "Hard" Satisfiability Problems Using GridSAT**, W. Chrabakh, R. Wolski, UCSB, USA, Pages 79-92.
- **Grid computing for energy exploration**, D. Bevc, S.E. Zarantonello, N. Kaushik, I. Musat, 3DGeo Development Inc. USA, Pages 93-106.
- **Air Quality Forecasting on Campus Grid Environment**, Y. Yan, B.M. Chapman, B. Sundaram, University of Houston, USA, Pages 107-119.
- **Experiences from Simulating the Global Carbon Cycle in a Grid Computing Environment**, J. Cope, C. Hartsough, S. McCreary, P. Thornton, H.M. Tufo, N. Wilhelmi, M. Woitaszek, University of Colorado and NCAR, USA. Pages 120-129.

Special Additional Presentation:

- **Grid Application Programming Environments: Comparing ProActive, Ibis, and GAT**, Thilo Kielmann, Andre Merzky, Henri Bal, Francoise Baude, Denis Caromel, Fabrice Huet, Vrije Universiteit, Netherlands and INRIA Sophia Antipolis, France, Pages 130-134.

6. Discussion Summaries

The following summaries are based on the notes taken in the sessions. They are not intended to provide certain outcomes or even consensus among the participants. They merely document the flow of arguments during the discussions.

6.1 Environments, Middleware, and Tools (moderated by Judith Utley)

The discussion was started with the following question.
Do grids help?

The key topic raised by participants was the need for user abstraction from the underlying fabric of the grid due to its complexity and hence ability to mystify and confuse users. This would be a very complicated task though and the amount of effort taken should be measured against the effect of not doing it. It was pointed out also that currently there are many hundreds of application users that have no idea about grid or whichever underlying infrastructure they are using.
One of the ideas for successful abstraction is the use of portals as they can hide many aspects of underlying functionality as an alternative to the command line. There have been cases though of GUI/Portals being developed initially and after trials having to develop a command line interface afterwards. It is accepted though that even with command line interfaces, the functionality must be high level. It must also be remembered that there are other user environments that are frequently requested, batch being the most popular though cycle scavenging is also catching on fast.

The next section of the discussion was started by the following question.
What is the most useful part/property of grids today?

The first and most obvious advantage provided by the grid is the additional computational power that is offered. A significant data transport capability though is rapidly becoming another key advantage as these systems become more and more the norm. This should become just a daily problem which has a solution. Data discovery of attributes is also very important, and will allow an additional new advantage which can give significant added value to generated data sets and hence work done. Another benefit highlighted of the grid is the possibility to have a large application that could in theory scale successfully across many systems. An example from the audience suggested that, for example, SDSC users generate large data volumes and an important capability would be data transfer tools which could make a huge difference.

These responses prompted the question,
Is grid technology expanding?

It was generally thought though, that there are still several areas that need work to ensure that the very slow uptake currently is accelerated. Current impediments were identified as:
- Hugely specialized administrative load and non-intuitive installations.
- Lack of an easy to use GUI framework to which plug-ins can be attached which can then help with integration of applications.
- The most deterring factor identified though was ease of use and user interfaces which are currently non-intuitive for non-grid people.

It was agreed that more documentation and/or tutorials would help though new/better/simpler technology would be preferable. A model of support described to get around these problems was described for moving users onto a general purpose grid setup. One example described was administrators contacting each user on a 1:1 basis and offering direct support to them, ensuring they experience minimal change, and the grid developer has a significant understanding of users' work currently and therefore what needs to be changed for successful grid usage.
The scalability of this solution was questioned though as you are left with grid educated users in the community who then spread the word and this hence creates more social acceptance of grid within currently skeptical research communities. The other advantage of this method of interaction is that the quality of the available user documentation increases through the snowball

effect. This could be summarized as simple usage backed up by social acceptance and good documentation. The other key point made is that if the grid is the only way that a problem could be tackled then it will get used. It was then asked whether the current method that all applications have individual schemes for fault tolerance was because there was nothing better available, or is application knowledge needed itself? The response was that this could be done by middleware, but is not always available. An example of middleware that handles failures well was given as Condor as well as Unicore.
The most promising way that the grid developer community could assist with the uptake of grid solutions though would be a fully integrated grid solution, i.e. all components that you need to create a grid, examples of which currently are GAT, (about 60% of necessary components), Condor (good though not really based upon common standards) and EGEE (though massively complicated and not ideal for small/medium applications).
It was pointed out that the application layer logic is missing (packaging, no coding, ...) and the largest impediment for users is that the current method of application assessment (will it work on a grid?) should be a lot easier!

6.2 Application Experiences (moderated by Thomas Hinke)

This discussion was started with the general question:
Why do applications go on the grid?

From a commercial point of view there are the normal reasons, easily extended computing power therefore increasing competitiveness and allowing the production of more viable software. This also allows significantly larger problems to be tackled due to the scalability of the solution. Therefore the most desirable solution is to construct a general solution for all of these problems in terms of workflows etc.
The other main reason given for use of the grid was particularly relevant to this problem as more and more complicated problems are being investigated and so workflows are increasingly necessary to perform the research. The ease of generalization could be increased if it was in designers minds, though it is not always feasible. A great problem currently though is the lack of generalized end-to-end solutions. Without this the approach you need a 'project' to get an application using the technology, which can be difficult due to funding constraints. These are two different things though, grid enabling existing apps and initial application development to go onto the grid. Diversity of projects make this difficult and so creation of the general solution should wait until specific solutions have been created. From a cost point of view of course the more general, the more cost efficient this becomes and since you could arrange for projects that look similar to use the same methods, therefore a toolkit could be developed of knowledge. The users though are most comfortable with a core API, but development is application-oriented then this can be tricky. Designing a set of tools, and using them though does not need to be the same set for everything. Interoperability of these tools though is the most important factor.
Looking at grids currently does present several different outward faces, some look like traditional clusters with added heterogeneity and hence putting apps onto these other environments seems particularly difficult with all applications presented. Creating standards is the easiest way to get around this. Using a portal or similar tool is also a way in which the underlying lack of standards can be hidden. Another problem highlighted was the significant difficulty of installation and configuration and solutions suggested were the use of a packaging toolkit. A suggestion for packaging is to use RPM. This though would need significant tailoring and is not platform independent.
One of the major components that are currently missing is accounting. This will become especially necessary within a commercial environment and increasingly in the academia and could be seen as a barrier to further uptake of grids. This should be the common layer and unless we have that we won’t see similarities developing. The other key service that was suggested that could drive commonality is file transfer.

Would it be possible to package a complete application to move when running?
As it is desirable to have tools available in as many locations as possible, would ‘package and move’ be viable? It was felt quite strongly that this was possible with the only restriction being a packaging service available on all resources. Generally, application users expect to be able to

move a complete environment. One of the only ways that this could be successfully done would be using virtualization of environments so that this could be stopped and started as well as ensuring that the presentation to the application developer could remain consistent on all platforms and locations.
The suggestion was made though that if all systems are located within a single financial domain then having a single policy for systems so you can install all your services on all your systems. This solution is sustainable but management becomes severely problematical the larger the installation. It was pointed out that this was the task of GGF, but its not there yet: should be able to rely on set of interfaces to be available.
An example given of the problems faced by grid users and developers though is the GridChem project They have found that they can't rely on all Chemistry departments to install GridChem software for their users. Another scenario that was described was "I need next week 1000 more CPUs, find some!" It was suggested that the chance that finding the right services could be very difficult though. The EU project GRASP was suggested as a possible way around this though the old issues of data transfer, architecture used etc. are still unresolved and hence subject to change. Eventually it does all come back to sociological pressures and trust. This cold be dealt with wider adoption. A key problem also though is that people don't follow standards. It was then asked can we enforce standards but pointed out that the GGF can only encourage.

What keeps applications from the grid?
It was suggested that grid is currently not simple enough, with no 'Dummys' book for grid and largely a lack of documentation and missing stability. The lack of stability has been one of the more publicized problems of the grid and, of course, a grid is useful only as long as it works! The major question was though is there a demand? It was thought that sure the demand was there, but you can't write the book yet... with the current standards process seeming to slow things down. It could be better if de-facto standards were made through the user community which could help, (this works in Java very well) but it must fulfill demand.
There are many applications which don't need Grid, but COULD use it and if we don't make it trivial for them, they won't use it. We must be careful though as there are some problems that can be solved by just buying a bigger computer which may end up actually being cheaper. Until this changes, it is difficult to see grid use expand massively. This does though hint at renting CPU power which grids promise. One scenario suggested was "I need 10.000 CPUs?" So it is necessary to be able to lookup in a registry: "who has 10.000 CPUs able to run GAUSSIAN". The overriding point where we are at currently though is that Grid software is not at a point where cost/benefit analysis can be applied. The other major headache that could prevent uptake is security as well with security requirements that are very diverse etc. When asked how many applications are using Grid now within the workshop, 10 people responded. It was pointed out that the SAGA group's work could assist, though a general measure of success was suggested of the size of projects could be a good metric, there will always be huge ones but the small ones (~two people) are getting more interesting: if you catch those, you are a success!

6.3 Final Discussion (moderated by Thilo Kielmann)

The final discussion quickly focused on the question: What is the killer application or feature? What will make my work worth doing? With several features given as examples such as replica data management and meta scheduling. It is essential that we use the good publicity that could be generated from these killer features. It was suggested though that grid was still a solution without a problem (vitamin versus aspirin) with several comments made that bearing in mind this is GGF14 there is still no clear idea what a grid is. Whenever new users come into the sphere they are faced instantly by a gigantic number of acronyms, this generates chaos. It is also currently obvious that there is no single killer application and this should be worked on so that decision makers (as several members of the audience were) become in favour of Grids? It was felt that we need to generate business models? One academic model suggested was that bearing in mind we currently have CampusGrid A, Campus Grid B, Campus Grid C... etc we really need a common definition of what a campus grid is though it is was suggested that just because there is a grid on campus it is not necessarily a campus grid. Hence it was proposed that we have more discussions and a further workshop on this.

7. Author Information

The bulk of this document consists of the individual, contributed papers. Their authors can be reached according to the information given in the respective papers. The overall volume has been edited by:

David Wallom
Campus Grid Manager
University of Oxford
13 Banbury Road
Oxford
ENGLAND
OX2 6NN

david.wallom@ierc.ox.ac.uk

Thilo Kielmann
Vrije Universiteit
Dept. of Computer Science
De Boelelaan 1083
1081HV Amsterdam
The Netherlands

kielmann@cs.vu.nl

8. Security Considerations

There may be security issues related to the individual solutions presented at the workshop. These need to be considered on a per-paper basis.

9. Intellectual Property Statement

10. Full Copyright Notice

Appendix: Papers contributed to the Workshop

The remainder of this document consists of the papers that had been accepted for presentation at the workshop.

Production-Quality Grid Environments with UNICORE

D. Erwin, M. Rambadt, A. Streit, and Ph. Wieder

Central Institute for Applied Mathematics (ZAM)
Forschungszentrum Jülich (FZJ)
52425 Jülich, Germany

Abstract. The UNICORE Grid technology provides a seamless, secure, and intuitive access to distributed Grid resources. Since its initial funding in two German-funded research projects, UNICORE evolved to a full-grown and well-tested Grid middleware system. Today it is used in daily production at many supercomputing centers.
In this paper we present an overview on the UNICORE production environments at the John von Neumann-Institute for Computing and within the European DEISA project.

1 Introduction

End of 1998 the concept of “Grid computing” was introduced in the monograph “The Grid: Blueprint for a New Computing Infrastructure” by I. Foster and C. Kesselman [7]. Almost two years earlier, in 1997, the development of UNICORE - Uniform Interface to Computing Resources - was initiated to enable German supercomputer centers to provide their users with a *seamless, secure, and intuitive access* to their heterogeneous computing resources. Like in the case of the Globus Toolkit® [4] UNICORE was started before “Grid Computing” became the accepted new paradigm for distributed computing. At the beginning UNICORE was developed as a prototype software in the two German funded projects UNICORE[1] [1] and UNICORE Plus[2] [2]. Over the following years, in various European-funded projects, UNICORE evolved to a full-grown and well-tested Grid middleware system, which today is used in daily production at many supercomputing centers worldwide and became a solid basis for research projects like EUROGRID, OpenMolGRID, UniGrids, and the Japanese NaReGI project. In this paper we present an overview on the usage of UNICORE for production in the John von Neumann-Institute for Computing (NIC) at the Research Center Jülich and in the European DEISA project.

The remainder of this paper is structured as follows. In Section 2 UNICORE's architecture and core features are described in more detail. Section 3 gives an overview on the usage of UNICORE in production and lessons learned. The paper ends with a brief conclusion.

2 The Architecture of UNICORE

Figure 1 shows the layered Grid architecture of UNICORE consisting of user, server and target system tier [11]. The implementation of all components shown is realized in Java. UNICORE meets the Open Grid Services Architecture (OGSA) [5] concept following the paradigm of 'Everything being a Service'. Indeed, an analysis has shown that the basic ideas behind UNICORE already realize this paradigm [13, 12].

[1] funded by BMBF grant 01 IR 703, duration: August 1997 - December 1999
[2] funded by BMBF grant 01 IR 001 A-D, duration: January 2000 - December 2002

2.1 User Tier

The UNICORE Client provides a graphical user interface to exploit the entire set of services offered by the underlying servers. The client communicates with the server tier by sending and receiving Abstract Job Objects (AJO) and file data via the UNICORE Protocol Layer (UPL) which is placed on top of the SSL protocol. The AJO is the realization of UNICORE's job model and central to UNICORE's philosophy of abstraction and seamlessness. It contains platform and site independent descriptions of computational and data related tasks, resource information and workflow specifications along with user and security information. AJOs are sent to the UNICORE Gateway in form of serialized and signed Java objects, followed by an optional stream of bytes if file data is to be transferred.

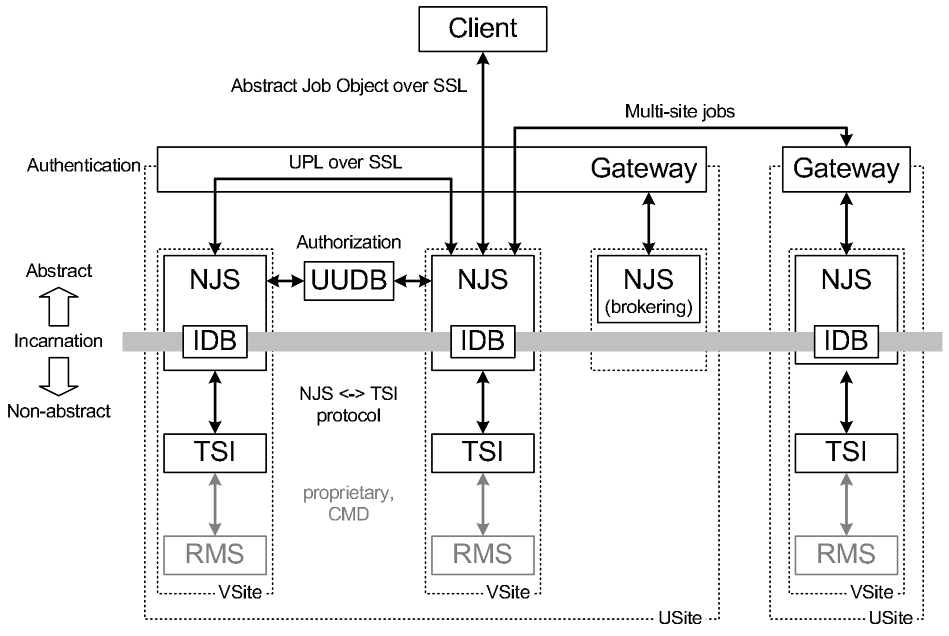


Fig. 1. The UNICORE architecture.

The UNICORE client assists the user in creating complex, interdependent jobs that can be executed on any UNICORE site (Usite) without requiring any modifications. A UNICORE job, more precisely a job group, may recursively contain other job groups and/or tasks and may also contain dependencies between job groups to generate job workflows. Besides the description of a job as a set of one or more directed a-cyclic graphs, conditional and repetitive execution of job groups or tasks are also included. For the monitoring of jobs, their status is available at each level of recursion down to the individual task. Detailed log information is available to analyze potential error conditions. At the end of the execution of the job it is possible to retrieve the stdout and stderr output of the job. Data management functions like import, export, and transfer are available through the GUI as explicit tasks. This allows the user to specify data transfer from one target system to another (e. g. for workflows), from or to the local workstation before or after the execution of a job, or to store data permanently in archives.

The previously described features already provide an effective tool to use resources of different computing centers both for capacity or capability computing, but many scientists and engineers use application packages. For applications without a graphical user interface, a tool kit simplifies the development of a custom built UNICORE plug-in. Over the years many plug-ins were developed, so that plug-ins already exist for many standard scientific applications, as e. g. for CPMD (Car-Parrinello Molecular Dynamics), Fluent, Gaussian, or MSC Nastran.

2.2 Server Tier

The server tier contains the Gateway and the Network Job Supervisor (NJS). The Gateway controls the access to a Usite and acts as the secure entry point accepting and authenticating UPL requests. A Usite identifies the participating organization (e. g. a supercomputing center) to the Grid with a symbolic name that resolves into the URL of the Gateway. An organization may be part of multiple Grids offering the same or different resources to different communities. The Gateway forwards incoming requests to the underlying Network Job Supervisor (NJS) of a virtual site (Vsite) for further processing. The NJS represents resources with a uniform user mapping scheme and no boundaries like firewalls between them.

A Vsite identifies a particular set of resources at a Usite and is controlled by a NJS. A Vsite may consist of a single supercomputer, e. g. a IBM p690 System with LoadLeveler, or a Linux cluster with PBS as resource management system. The flexibility of this concept supports different system architectures and gives the organization full control over its resources. Note that there can be more than one Vsite inside each Usite as depicted in Figure 1.

The NJS is responsible for the virtualization of the underlying resources by mapping the abstract job on a specific target system. This process is called "incarnation" and makes use of the Incarnation Database (IDB). System-specific data are stored in the IDB describing the software and hardware infrastructure of the target system. Among others, the available resources like software, incarnation of abstract commands (standard UNIX commands like rm, cp, ...) and site-specific administrative information are stored. In addition to the incarnation the NJS processes workflow descriptions included in an AJO, performs pre- and post-staging of files and authorizes the user via the UNICORE User Database (UUDB). Typically the Gateway and NJS are running on dedicated secure systems behind a firewall, although the Gateway could be placed outside a firewall or in a demilitarized zone.

2.3 Target System Tier

The Target System Interface (TSI) implements the interface to the underlying supercomputer with its resource management system. It is a stateless daemon running on the target system and interfacing with the local resource manager realized either by a batch system like PBS, a batch system emulation on top of e. g. Linux, or a Grid resource manager like Globus' GRAM [6, 9].

2.4 Single Sign-On

The UNICORE security model relies on the usage of permanent X.509 certificates issued by a trusted Certification Authority (CA) and SSL based communication across 'insecure' networks. Certificates are used to provide a single sign-on in the client. The client unlocks the user's keystore when it is first started, so that no further password requests are handed to the user. All authentication and authorization is done on the basis of the user certificate. At each UNICORE site user certificates are mapped to local accounts (standard UNIX uid/gid), which may be different at each site, due to existing naming conventions. The sites retain full control over the acceptance of users based on the identity of the individual – the distinguished name – or other information that might be contained in the certificate. UNICORE can handle multiple user certificates, i. e. it permits a client to be part of multiple, disjoint Grids. It is also possible to specify project accounts in the client allowing users to select different accounts for different projects on one execution system or to assume different roles with different privileges.

The private key in the certificate is used to sign each job and all included sub-jobs during the transit from the client to sites and between sites. This protects against tampering while the job is transmitted over insecure internet connections and it allows to verify the identity of the owner at the receiving end, without having to trust the intermediate sites which forwarded the job.

3 UNICORE-Based Production Environments

3.1 Production System on Jump

Since July 2004 UNICORE is established as production software to access the supercomputer resources of the John von Neumann-Institute for Computing (NIC) at the Research Center Jülich. These are the 1312-processor IBM p690 cluster (Jump) [8], the Cray SV1 vector machine, and a new Cray XD1 cluster system. As an alternative to the standard SSH login, UNICORE provides an intuitive and easy way for submitting batch jobs to the systems. The academic and industrial users come from all over Germany and from parts of Europe. The applications come from a broad field of domains, e. g. astrophysics, quantumphysics, medicine, biology, chemistry, and climate research, just to name the largest user communities. In the first five month of 2005 31.51% of the used CPU-cycles of the Jump system where used by UNICORE jobs, details for each month are in Table 1.

month	used CPU-cycles of UNICORE jobs
January 2005	30.45%
February 2005	30.50%
March 2005	27.07%
April 2005	29.74%
May 2005	39.13%

Table 1. Usage Statistics of UNICORE.

A dedicated, pre-configured UNICORE client with all required certificates and accessible Vsites is available for download. This alleviates the installation and configuration process significantly. Furthermore, an online installation guide including a certificate assistant, an user manual, and example jobs help users getting started.

To provide the NIC-users with adequate certificates and to ease the process of requesting and receiving a certificate, a certificate authority (CA) was established. User certificate requests are generated in the client and have to be send to the CA. Since introduction of UNICORE at NIC, more than 120 active users requested a UNICORE user certificate.

A mailing list serves as a direct link of the users to UNICORE developers in the Research Center Jülich. The list allows to post problems, bug reports, and feature requests. This input is helpful in enhancing UNICORE with new features and services, in solving problems, identifying and correcting bugs, and influences new releases of UNICORE available at SourceForge [15].

3.2 DEISA – Distributed European Infrastructure for Scientific Applications

Traditionally, the provision of high performance computing resources to researchers has been the objective and mission of national HPC centers. On the one hand, there is an increasing global competition between Europe, USA, and Japan with growing demands for compute resources at the highest performance level, and on the other hand stagnant or even shrinking budgets. To stay competitive major investments are needed every two years – an innovation cycle that even the most prosperous countries have difficulties to fund.

To advance science in Europe, eight leading European HPC centers devised an innovative strategy to build a Distributed European Infrastructure for Scientific Applications (DEISA)[3] [3]. The centers join in building and operating a tera-scale supercomputing facility. This becomes possible through deep integration of existing national high-end platforms, tightly coupled by a dedicated network and supported by innovative system and Grid software. The resulting virtual distributed supercomputer has the capability for natural growth in all dimensions without singular procurements at the European level. Advances in network technology and the resulting increase in bandwidth and lower latency virtually shrink the distance between the nodes in the distributed super-cluster. Furthermore, DEISA can expand horizontally by adding new systems, new architectures, and new partners thus increasing the capabilities and attractiveness of the infrastructure in a non-disruptive way.

By using the UNICORE technology, the four core partners of the projects have coupled their systems using virtually dedicated 1 Gbit/s connections. All other sites will follow in the next step. The DEISA super-cluster currently consists of over 4000 IBM Power 4 processors and 416 SGI processors with an aggregated peak performance of about 22 teraflops. UNICORE provides the seamless, secure and intuitive access to the super-cluster.

The Research Center Jülich is one of the DEISA core partners and is responsible for introducing UNICORE as Grid middleware at all partner sites and for providing support to local UNICORE administrators.

In the following we describe the DEISA architecture. Note, a detailed description of UNICORE's architecture and server components can be found in Section 2 and in particular in Figure 1. All DEISA partners have installed the UNICORE server components Gateway, NJS, TSI, and UUDB to access the local supercomputer resources of each site via UNICORE. Figure 2 shows the DEISA UNICORE configuration of the core production environment. For clarity only four sites are shown. At each site, a Gateway exists as an access to the DEISA infrastructure. The NJSs are not only registered to their local Gateway, but to all other Gateways at the partner sites as well. Local security measures like firewall configurations need to consider this, by permitting access to all DEISA users and NJSs. This fully connected architecture has several advantages. If one Gateway has a high load, access to the high performance supercomputers through DEISA is not limited. Due to the fully connected architecture, no single point of failure exists and the flexibility is increased.

The DEISA partners operate different supercomputer architectures, which are all accessible through UNICORE. Initially all partners with IBM p690 clusters are connected to one large virtual

[3] funded by EC grant FP6-508803, duration: May 2004 - April 2009

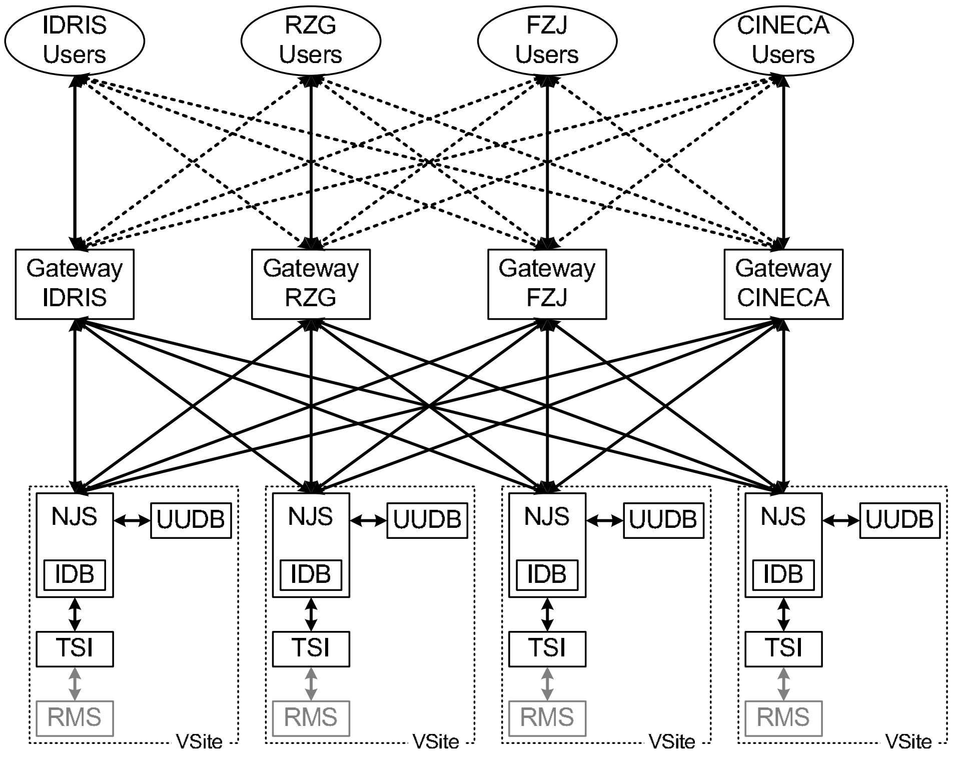


Fig. 2. The DEISA architecture of the core production environment.

supercomputer. In a second step other supercomputers of different variety are connected to DEISA, making the virtual supercomputer heterogeneous. UNICORE can handle this, as it is designed to serve such heterogeneous architectures in a seamless, secure, and intuitive way.

In December 2004 a first successful UNICORE demonstration between the four DEISA core sites FZJ (Research Center Jülich, Germany), RZG (Computing Center Garching, Germany), CINECA (Italian Interuniversity Consortium, Italy) and IDRIS (Institute for Development and Resources in Intensive Scientific Computing, France) was given. Different parts of a distributed astrophysical application were generated and submitted with UNICORE to all four sites.

The experience and knowledge of the researchers, developers, users, and administrators in working with UNICORE in the DEISA project on a large production platform will be used as useful input for future developments of the UNICORE technology. A close synchronization with the UniGrids project [16] is foreseen.

3.3 Lessons Learned

The deployment of new software to be used in production has to offer added value to the users, otherwise the new software will not be accepted. Hence, users have to be stimulated to use the software. They have to be encouraged to use Grid technology for applications, computations, data transfers, and access to resources, to make their applications Grid-aware, and to consider Grid technology for their daily problem solving.

The obvious prerequisite is that the software provides the necessary functions users require. However, the beginning of the UNICORE development showed that the sole fulfillment of these requirements if not sufficient. Although all requested functions were present, only a small number of users used UNICORE for their daily work.

The transition from prototype to production requires:

– High quality of the software, especially high reliability and resilience.
– Help for the users to overcome initial hurdles and permanent assistance to users in case of problems.
– Permanent, 24/7 availability of the infrastructure.
– A long term commitment for continuous development and support.

As soon as the Research Center Jülich implemented and gave the commitment for these additional aspects through initiating and supporting the UNICORE@SourceForge [15] initiative, the use of UNICORE increased significantly as the statistics confirm.

UNICORE and Grid technology in general have made the step away from being used just for demos towards real work in production environments. The driving forces for this effect in our center and the DEISA infrastructure are the above described features of the technology. Through the single sign-on mechanism users do no longer have to remember different account names and passwords for different machines and projects. The easy-to-use and intuitive graphical client allows users to define complex workflows with data staging, data- or time-dependencies between sub-jobs and involving different applications in a more comfortable and sophisticated way as with previously used tools of the local resource management system. Specifying data dependencies, both data-parallel and data-sequential jobs can be executed, enabling e. g. workflows of applications, where the output of one step is used as input for the next step. According to user feedback this feature is very valuable for applications from quantum computing and bio-molecular science. Once jobs are fully defined including an appropriate resource assignment, they can be saved for later usage. By re-loading previously stored jobs in the client, users are able to submit one specific job or workflow multiple times, only with e. g. different input data or input parameters in order to conduct parameter studies.

A detailed analysis of the jobs shows that the increase in resource consumption of UNICORE jobs (cf. Table 1) is due to the submission of very large jobs through UNICORE. These jobs are both large in CPU requirements as well as in run time. Typically the resources are used for parameter studies with simulation codes. It can be observed that users combine several iterations and steps (about 20 to 30) within a single UNICORE workflow with data- and/or time-dependencies.

Several projects from various domains of science make use of UNICORE to access the compute resources at Research Center Jülich. Among these projects are:

– QCD simulations with light quark flavors (elementary particle physics)
– Finite temperature meson correlation functions (elementary particle physics)
– Non-leptonic kaon decays in lattice QCD (elementary particle physics)
– Nucleon matrix elements from overlap fermions (elementary particle physics)
– Electronic and optical properties of capped silicon and germanium nanocrystallites (material science)
– Visco-elastic shear flow (fluid science)
– Small scale structure of the universe (astro physics)

4 Conclusion

In this paper we presented the usage of UNICORE in production-quality Grid environments. UNICORE – Uniform Interface to Computing Resources – provides a *seamless, secure and intuitive* access to distributed Grid resources. Initially developed in two German projects, the software evolved from prototype status to a full-grown and well-tested Grid system, which is today used in daily production at many supercomputing centers in Europe.

At the John von Neumann-Institute for Computing, Research Center Jülich, many users submit their batch jobs through UNICORE to the 1312-processor 8.9 TFlop/s IBM p690 cluster, the Cray SV1 vector machine, and a new Cray XD1 cluster system. Leading European HPC centers joined in the project DEISA to build and operate a distributed European Grid supercomputing infrastructure for scientific applications with multi tera-scale performance and production quality, similar to the TeraGrid project [14]. Within the DEISA project UNICORE is used as the Grid middleware to connect all sites to the infrastructure.

The future of UNICORE is promising and follows the trend of "Everything being a Service", as it follows the Open Grid Service Architecture (OGSA) [5]. In this context, the UniGrids[4] project [16] continues previous efforts in integrating the Web Services and UNICORE technology to enhance UNICORE to an architecture of loosely-coupled components while keeping its "end-to-end" nature. To this end UNICORE/GS will be developed, which makes UNICORE compliant with the Web Services Resource Framework (WS-RF) [10].

Since May 2004 the UNICORE software is available as open source under a BSD licence from SourceForge for download [15]. Numerous contributors from all over the world, e. g. Norway, Poland, China and Russia checked-in their developments and until April 2005 more than 3100 downloads of UNICORE were counted.

References

1. D. Erwin (Ed.). *UNICORE - Uniformes Interface für Computing Ressourcen, final project report (in German)*. 2000.
2. D. Erwin (Ed.). *UNICORE Plus Final Report - Uniform Interface to Computing Resources*. Forschungszentrum Jülich, 2003.
3. DEISA - Distributed European Infrastructure for Supercomputing Applications. `http://www.deisa.org`.
4. I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal on Supercomputer Applications*, 11(2):115–128, 1997.
5. I. Foster, C. Kesselmann, J. M. Nick, and S. Tuecke. The Physiology of the Grid. In F. Berman, G. C. Fox, and A. J. G. Hey, editors, *Grid Computing*, pages 217–249. John Wiley & Sons Ltd, 2003.
6. Globus: Research in Resource Management. `http://www.globus.org/research/resource-management.html`.
7. I. Foster, C. Kesselman (Eds.). *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc. San Fransisco, 1999.
8. Jump - Juelich Multi Processor, 8,9 TFlop/s IBM p690 eServer Cluster. `http://jumpdoc.fz-juelich.de`.
9. R. Menday and Ph. Wieder. GRIP: The Evolution of UNICORE towards a Service-Oriented Grid. In *Proc. of the 3rd Cracow Grid Workshop (CGW'03)*, pages 142–150, 2003.
10. OASIS Web Services Resource Framework (WSRF). `http://www.oasis-open.org/committees/wsrf`.
11. M. Romberg. The UNICORE Grid Infrastructure. *Scientific Programming*, 10(2):149–157, 2002.
12. D. Snelling. UNICORE and the Open Grid Services Architecture. In F. Berman, G. Fox, and T. Hey, editor, *Grid Computing: Making The Global Infrastructure a Reality*, pages 701–712. John Wiley & Sons, 2003.
13. D. Snelling, S. van den Berghe, G. von Laszweski, Ph. Wieder, D. Breuer, J. MacLaren, D. Nicole, and H.-Ch. Hoppe. A UNICORE Globus Interoperability Layer. *Computing and Informatics*, 21:399–411, 2002.
14. TeraGrid. `http://www.teragrid.org/`.
15. UNICORE at SourceForge. `http://unicore.sourceforge.net`.
16. UniGrids - Uniform Access to Grid Services. `http://www.unigrids.org`.

[4] funded by EC grant IST-2002-004279, duration: July 2004 - June 2006

Streamlining Grid Operations: Definition and Deployment of a Portal-based User Registration Service

Ian Foster[1,2] Veronika Nefedova[1] Lee Liming[1] Rachana Ananthakrishnan[1]
Ravi Madduri[1] Laura Pearlman[3] Olle Mulmo[4] Mehran Ahsant[4]

[1] Argonne National Laboratory, Argonne, IL, 60439
[2] University of Chicago, Chicago, IL, 60637
[3] Information Sciences Institute, University of South California, Marina del Rey, CA 90292
[4] Kungliga Tekniska Högskolan, Nada, 100 44 Stockholm, Sweden

Abstract

Manual management of public key credentials can be a significant and often off-putting obstacle to Grid use, particularly for casual users. We describe the Portal-based User Registration Service (PURSE), a set of tools for automating user registration, credential creation, and credential management tasks. PURSE provides the system developer with a set of customizable components, suitable for portal integration, that can be used to address the full lifecycle of Grid credential management. We describe the PURSE design and describe how it has been used within portals for two different systems, the Earth System Grid data access system and the Swegrid computational grid. In both cases, the user is entirely freed from the need to create or manage public key credentials, thus simplifying their Grid experience and reducing opportunities for error. We argue that this capturing of common use cases in a reusable "solution" can be a model for how Grid ease-of-use can be addressed in other domains as well.

1 Introduction

A typical Grid application requires that a set of users share resources of various kinds in some controlled manner. To this end, many extant Grid deployments use the public-key infrastructure (PKI)-based Grid Security Infrastructure (GSI) [10] as a basis for secure user single sign on and subsequent authentication of users and resources prior to authorization. GSI defines and implements useful algorithms for authentication and delegation. However, the tasks of creating and managing the PKI credentials used by GSI can be significant sources of complexity, user difficulty, and even error (and thus insecurity) in Grid deployments.

These considerations motivate our design of the Portal-based User Registration Service (PURSE), a set of tools for developing portal-based systems that automate user registration, the creation of PKI credentials, and subsequent credential management. A typical PURSE-based portal allows a user to register via a Web page, follow which a credential is created and managed on their behalf, with subsequent access provided via a username and password. A separate administrator interface allows a portal administrator to approve requests, revoke credentials, and so forth. By streamlining and codifying these various steps, PURSE-based systems can significantly reduce barriers to the integration of new users, overheads associated with credential management, and opportunities for error—and thus simplify the development of usable Grid applications.

An important PURSE design goal was to support the creation and use of PKI credentials of varying "quality." It is often the case that different access control policies are associated with different resources and operations. For example, some operations and resources (e.g., write access to archival storage) may require stringent verification of the identity and/or attributes of a requestor, while others (e.g., read access to Web pages) require only audit of a weakly authenticated identity. The definition and enforcement of such policies can be a significant source of complexity in Grid application deployments, due to the need not only to implement policies correctly but also to achieve appropriate tradeoffs between operational security and ease of use. Thus, PURSE mechanisms allow for the automatic creation of credentials following either simple online registration or stringent identity verification, and for the upload of existing credentials.

The PURSE implementation is not particularly complex, being based on an integration of a number of existing components, including GSI libraries, the MyProxy online credential repository, the SimpleCA credential generator, and portal tools. This implementation approach of integrating existing components to construct a reusable "solution" that addresses an important set of use cases is one that we hope will be pursued by many other Grid developers.

We have recently become aware of the Grid Account Management Architecture (GAMA) project [1, 9], which has produced similar mechanisms in parallel with our PURSE development. GAMA differs from PURSE in various mostly minor respects: for example, it is hosted on GridSphere rather than Axis, and does not support uploading of existing credentials, a critical requirement for all PURSE users to date. We view this parallel evolution as demonstrating the importance of this technology.

The rest of this paper describes in turn the PURSE system (Section 2), two different PURSE-based portals (Section 3), and the sample registration portal distributed with PURSE (Section 4). We conclude in Section 5.

2 System Description

The PURSE user registration system is a collection of Java APIs designed to work as a backend for a front-end user interface, typically a web portal, to ease registration and credential management. Driven by user requests through the interface, this Java code stores user contact information, generates and stores new credentials for users, and allows for subsequent use of those credentials to access Grid resources. The system has functionality to support credential renewal and revocation. This functionality can be accessed through a well-defined API and is easily configurable.

The system is built upon some common tools, as follows:

- A JDBC-compliant database is used to persist user data. (MySQL is currently used.)
- A Certification Authority is used to generate and sign user credentials. Depending on application requirements, either SimpleCA [6] or an external CA can be used for generating and signing users credentials.
- The MyProxy server [3, 11] is used to store user credentials
- JavaMail [2] is used to send and receive notifications to the user and CA operator.

2.1 Typical Usage Scenarios

A PURSE user must first register with the PURSE system. This is a one-time event that must precede any other use of the system. Registration involves three principal steps, as follows.

1. The user accesses the registration page on the portal and enters relevant information (e.g., contact information, desired user name, desired password).

2. PURSE persists the user information and, using the provided contact information, sends an email back to the requesting user requesting that they confirm the request. This email typically provides a link that the user can click to confirm the request. This step helps to prevent registration errors and to verify the legitimacy of the email address.
3. Upon confirmation, the submitted request is sent to the certificate authority (CA) configured in the PURSE system. The CA operator reviews the information provided by the user, checks the contact information and decides whether to approve or reject the request based on criteria of their choosing. If the request is rejected, an email is sent to the user notifying them of the decision. If the request is approved, then PURSE generates and stores long-term user credential in the MyProxy server. An email is then sent to the user notifying them that registration has completed successfully.

In a variant of this scenario, the user may instead supply an existing credential during the registration process. The same registration and approval process is followed, but following approval by the CA operator, the user is instructed to upload their existing certificate into the PURSE MyProxy.

Following successful registration, the user can use the username and password requested during registration to log in to the portal. The portal then retrieve a short-term credential for the user from the MyProxy service and uses that credential on behalf of the user to access VO resources as directed by VO-specific logic in the portal.

The overall system architecture is presented on Figure 1.

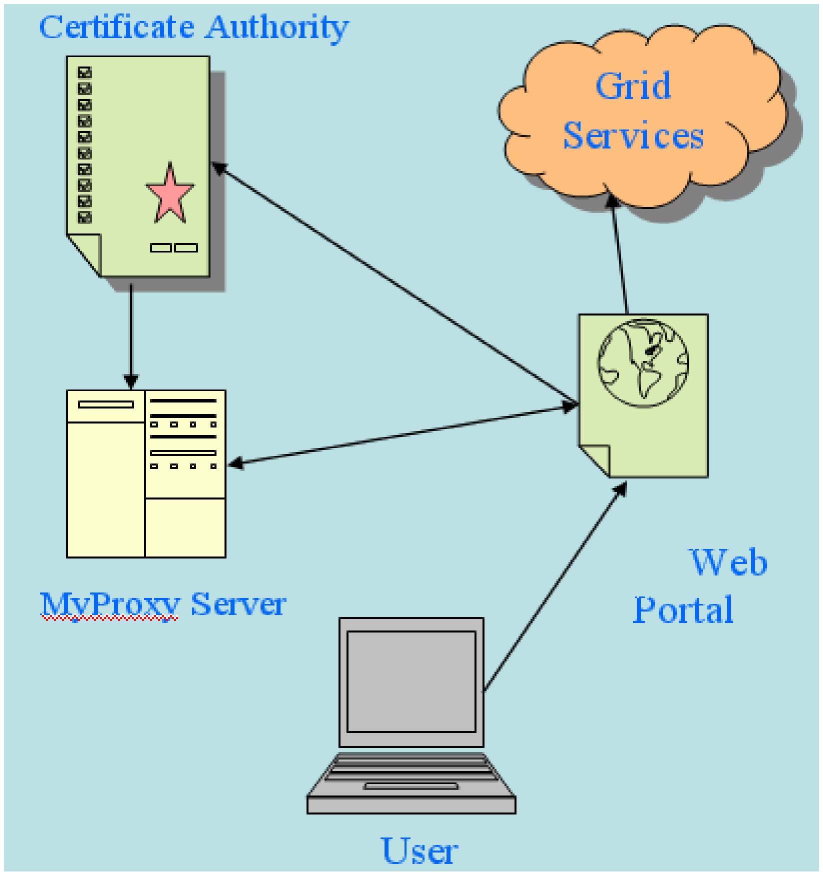


Figure 1: Sample Registration Portal Architecture

2.2 Overview of Registration System APIs

PURSE is structured as a set of building blocks that can be used to create a fully functional web-based portal for accessing the Grid. The modules are available as "jar" files and can be plugged into any front-end interface such as an existing portal. We describe the high-level functionality and APIs for these building blocks in the following.

New user registration

Register user: This step initiates user registration by storing relevant user information, including requested username and user email address in the backend database. Once the information is stored, an email is sent to the user requesting confirmation of request.

Process user request: This step is triggered by the user's confirmation of the request to the registration system. An email is sent to a configured CA operator email address with instructions for the CA operator to access the user details.

Accept user: This module is invoked when a CA operator accepts a particular user's request. The following steps are performed.

- If the user wishes to use their own credentials (from an outside CA), the user is sent an email with a link that, when clicked by the user, downloads a simple java MyProxy client using Java Webstart that the user can use to upload their credential to the PURSE MyProxy server.
- If the user does not have their own credentials:
    - Either SimpleCA is used to generate a certificate for the user or a certificate request is sent to an external CA, depending on application requirements.
    - Either the configured SimpleCA certificate is used to sign the certificate or a signed certificate is received from the external CA.
    - The resulting long-term credentials are loaded onto a MyProxy server.
    - The database is updated to set the user's request status to "accepted."
- In both cases, an email is sent to the user indicating that registration is complete.

Reject user: If the CA chooses to reject the user, this module is invoked. It sends an email to the user and updates the user request status to "rejected."

Managing registered user

Revoke user: This module deletes the user from registration system. The user's credentials are removed from the MyProxy server and the user's status in the database is set to "revoked."

Renewal notice: This operation can be run as a periodic task to send mail to all users whose credentials are due to expire in some configured timeframe.

Renew user: This operation is triggered by a user attempting to renew membership and sets the user status in the database to "renew." If the renewal request is granted, an API to generate new long term credentials for the user and store them in the MyProxy server is provided.

Tools for registered users

Change password: Allows a registered user to change their password.

2.3 PURSE Setup

Establishing a PURSE-based portal involves two steps. In the first, we develop the portal code or alternatively integrate PURSE calls into an existing portal. In the second step, we set up the backend database used to maintain user information (e.g., MySQL), a SimpleCA certificate authority (or alternatively configure PURSE to access an existing CA), and the MyProxy server used to store user credentials. Complete instructions for PURSE installation and testing are on the PURSE web side [5].

3 Deployment Use Cases

We describe two production deployments that have served both to drive PURSE requirements and to validate PURSE functionality.

3.1 Earth Systems Grid

PURSE was initially developed for the Earth System Grid (ESG) [8], a U.S. Department of Energy project to provide online access to climate data. We describe here the ESG production deployment as an example of how the registration system can be used. The following details are specific to deploying the Registration System for ESG.

The ESG portal needs to support two different classes of users: a small number of "privileged" users who can access all ESG data , including the newest data produced by the climate models, and all other users, who can access only publicly available, previously published data. Privileged users must be strongly authenticated, while for all other users, the requirement is to have some weak verification of their identity for the purpose of tracking ESG usage. At the same time, all users must have valid GSI credentials in order to access the data stored on the ESG various storage systems, which include NCAR MSS, NERSC HPSS, and GridFTP servers throughout the ESG grid.

This combination of authentication and authorization requirements spurred the development of PURSE. The user registration process plus email verification provides sufficient verification of user identity to satisfy requirements for tracking ESG usage. The ability to upload an existing credential supports the stronger authentication required for privileged users, who can obtain that credential from a CA with a stronger authentication policy. Users are then assigned to the appropriate user groups during registration, based on ESG policy. When a user would like to access data via the ESG Portal, their request is validated by the portal, which bases its decision on the user's group assignment. The number of user groups is configurable and depends on ESG policy. ESG uses the standard workflow for user registration, described in Section 2.1. ESG has 700 registered users as of May 2005.

Users who wish to see PURSE in action can register with the ESG portal by following the Registration link from the main ESG site (https://www.earthsystemgrid.org). At the registration web page, specify in the "Statement of Work" that you are interested in seeing PURSE in action. Access will be granted with limited access to ESG data.

3.2 Swegrid

Swegrid [7], a distributed computational resource in Sweden, uses the PURSE libraries to provides a registration system for its users. This system uses PURSE to meet Swegrid requirements for providing users with a certificate signed by an external (real) CA.

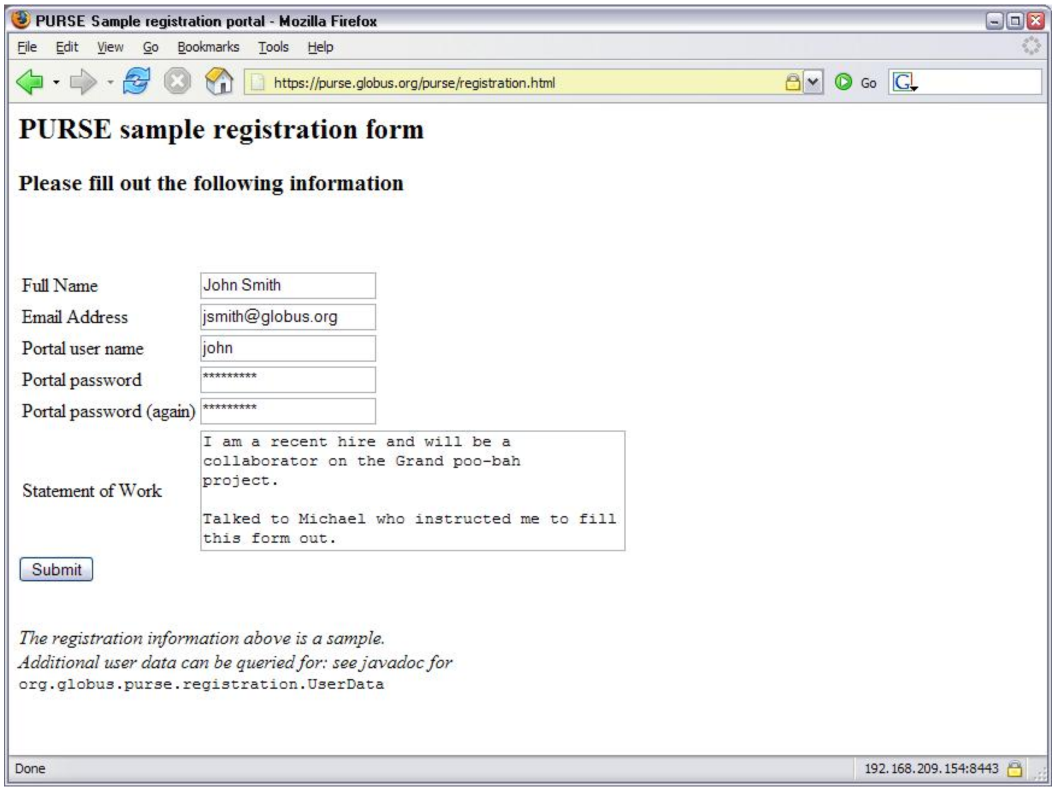
The main difference between the Swegrid and ESG registration system is the workflow for issuing certificates. In contrast to ESG, the Swegrid portal after registering the user in its local database sends a notification to the Swegrid registration authority which contains a link, which can be used by the RA to validate the user's information and to verify their identity against the papers signed and sent by that user. Upon approving the identity of the user, the RA sends the confirmation message to the Swegrid portal by replying to the notification email. The portal then accepts the user and generates a certificate request, which will be sent to the configured external and trusted certification authority. The CA may also use a similar link to access the local information saved on portal database in order to verify the user's identity. Upon approval, CA signs the certificate and sends it back to the Swegrid portal. The portal receives the signed certificate from the CA and uploads this to the MyProxy server.

4 Sample Portal User Registration Interface

The PURSE distribution includes code for a simple Sample Registration Portal that may be adapted to meet specific application requirements. The Sample Registration Portal solicits basic data from the user, generates a certificate request to the VO operator, (following approval) generates a certificate and stores it in the MyProxy server, and gives the user an identifier and password for MyProxy access. A separate administrator interface allows a CA operator to accept or reject user requests and also to revoke issued certificates.

User registration involves the following steps.

1. The user fills in the Sample Registration Portal’s entry page, shown in Figure 2, to submit their registration request.
2. The Sample Registration Portal verifies the user’s email by sending the mail in Figure 3(a) to the provided email address.
3. Following user acknowledgement, the CA operator receives an email notification when a new account is being requested, as in Figure 3(b).
4. After receiving this notification, the CA operator logs in to a secure web site (Figure 4) and views the request.
5. After the user’s credentials are generated and uploaded into MyProxy the user receives an email notification, as in Figure 3(c).


PURSE Sample registration portal - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://purse.globus.org/purse/registration.html
Go
Submit
Done
192.168.209.154:8443

PURSE sample registration form

Please fill out the following information

Full Name	John Smith
Email Address	jsmith@globus.org
Portal user name	john
Portal password	*********
Portal password (again)	*********
Statement of Work	I am a recent hire and will be a collaborator on the Grand poo-bah project. Talked to Michael who instructed me to fill this form out.

The registration information above is a sample.
Additional user data can be queried for: see javadoc for `org.globus.purse.registration.UserData`

Figure 2: Screenshot of the PURSE sample user registration interface

(a) Email confirmation step: message sent to user

Date: Thu, 1 Jul 2004 14:25:47 -0600 (MDT)
From: esgport@ucar.edu
To: john_smart@ucar.edu
Subject: ESG Registration

The Earth System Grid (ESG) Portal received a request for a new user account that uses your email address. Click on the link below to confirm your request (NOTE: you will not be able to login until you receive an email from the portal administrator indicating your request has been approved):

Hhttp://www.earthsystemgrid.org/security/confirmRequest.do?token=000000fd-7c62-605c-ffffdea0-766ad9819840H

If you did not request this account, please inform us at esg-admin@earthsystemgrid.org.

Thank you,

ESG System Administrator

(b) Email sent to CA operator for approval

From: esgport@ucar.edu
Date: July 1, 2004 12:17:07 AM MDT
To: esg-ca@ucar.edu
Subject: ESG Registration

A request has been made for user account on the ESG Portal. You may access the details of the request by clicking on the following link.

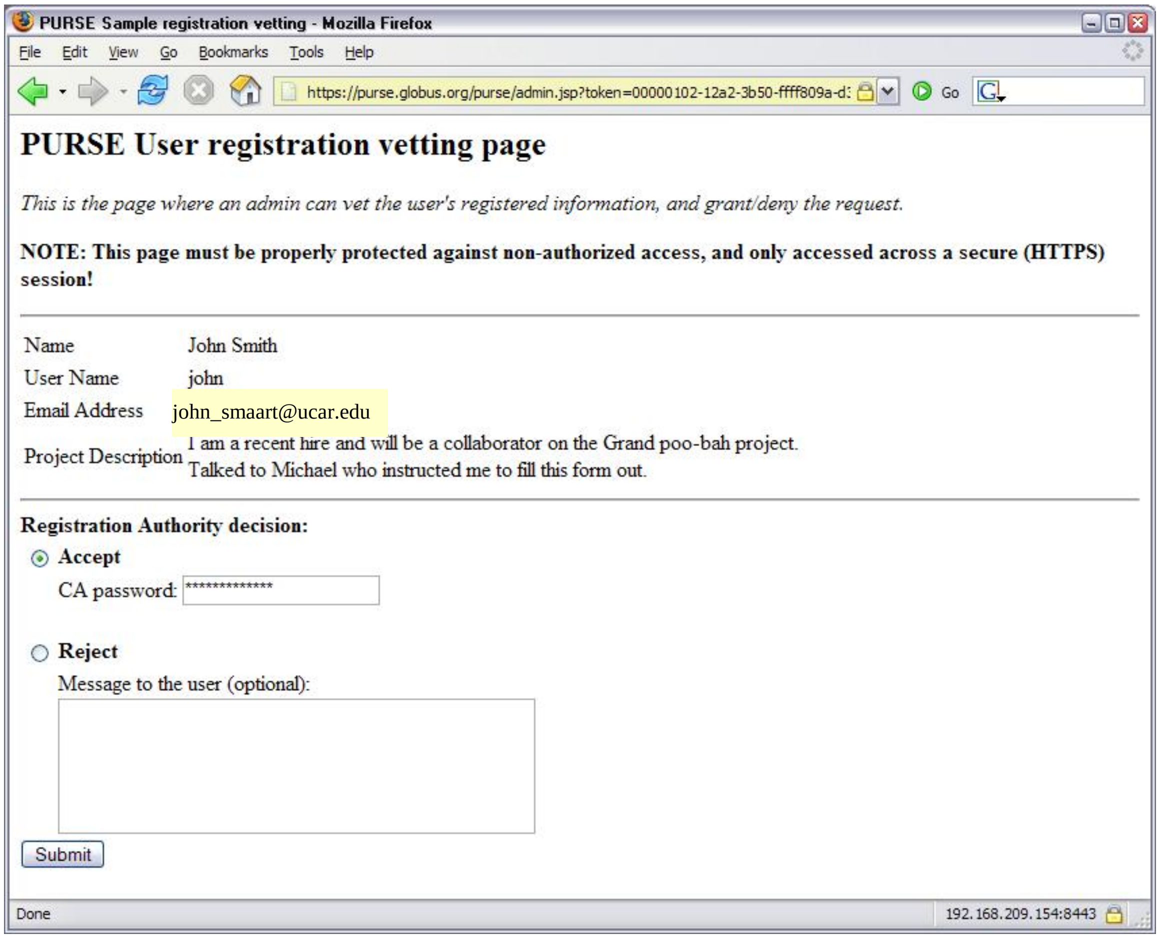
Hhttp://www.earthsystemgrid.org/administration/accountRequestData.do?token=000000fd-2e0e-5d33-00006ac0-8387f64897beH

(c) Registration confirmation email sent to user

Date: Thu, 1 Jul 2004 14:34:52 -0600 (MDT)
From: esgport@ucar.edu
To:john_smart@ucar.edu
Subject: ESG Registration

Your request for an account with the ESG portal has been approved.

Figure 3: The three emails sent during user registration (based on ESG operational system)


PURSE Sample registration vetting - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
https://purse.globus.org/purse/admin.jsp?token=00000102-12a2-3b50-ffff809a-d:
Go

PURSE User registration vetting page

This is the page where an admin can vet the user's registered information, and grant/deny the request.

NOTE: This page must be properly protected against non-authorized access, and only accessed across a secure (HTTPS) session!

Name	John Smith
User Name	john
Email Address	john_smaart@ucar.edu
Project Description	I am a recent hire and will be a collaborator on the Grand poo-bah project. Talked to Michael who instructed me to fill this form out.

Registration Authority decision:

[✓ **Accept**]

CA password: *************

Reject

Message to the user (optional):

Figure 4: Screenshot of the PURSE sample administrative interface

5 Summary and Next Steps

PURSE provides a set of tools that can be used to construct Web-based user and administrative interfaces for user registration, credential management, and Grid access. PURSE automates the process of obtaining PKI credentials for users; provides for the secure storage of credentials; allows users to use existing Grid credentials, if available; and provides for Grid access via Web portals and secure username-password authentication.

In future releases, we plan to work towards simplifying PURSE installation by creating an easy packaging solution for this system. In addition, we need to adapt the current implementation to separate the credential repository from the rest of the portal logic, so as to permit hosting of the credential repository on a secure system.

Acknowledgements

PURSE was first developed at Argonne National Laboratory in collaboration with the Earth Systems Grid, with support from the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract W-31-109-Eng-38. Staff at NCSA and the University of Chicago supported by NSF Middleware Initiative's GRIDS Center [4] contributed to the extensive pre-release testing. KTH provided the sample portal interface and added several features to the API.

References

1. Grid Account Management Architecture (GAMA), 2005. http://grid-devel.sdsc.edu/gamaT.
2. JavaMail, 2005. http://java.sun.com/products/javamail.
3. MyProxy, 2005. http://grid.ncsa.uiuc.edu/myproxy.
4. NSF Middleware Initiative (NMI) Grid Research Integration Development and Support (GRIDS) Center, 2005. www.grids-center.org.
5. Portal-based User Registration Service (PURSE), 2005. www.grids-center.org/solutions/purse.
6. SimpleCA, 2005. www.globus.org/security/simple-ca.html.
7. Swegrid, 2005. www.swegrid.se.
8. Bernholdt, D., Bharathi, S., Brown, D., Chanchio, K., Chen, M., Chervenak, A., Cinquini, L., Drach, B., Foster, I., Fox, P., Garcia, J., Kesselman, C., Markel, R., Middleton, D., Nefedova, V., Pouchard, L., Shoshani, A., Sim, A., Strand, G. and Williams, D. The Earth System Grid: Supporting the Next Generation of Climate Modeling Research. *Proceedings of the IEEE*, *93* (3). 485-495. 2005.
9. Bhatia, K., Lin, A., Link, B., Mueller, K. and Chandra, S. Geon/Telescience Security Infrastructure. San Diego Supercomputer Center, Technical Report TR-2004-5, 2004.
10. Foster, I., Kesselman, C., Tsudik, G. and Tuecke, S., A Security Architecture for Computational Grids. *5th ACM Conference on Computer and Communications Security*, 1998, 83-91.
11. Novotny, J., Tuecke, S. and Welch, V., An Online Credential Repository for the Grid: MyProxy. *10th IEEE International Symposium on High Performance Distributed Computing*, San Francisco, 2001, IEEE Computer Society Press.

Early Application Experience with the Grid Application Toolkit (GAT)

Stevens Le Blond, Ana-Maria Oprescu, Chen Zhang
Vrije Universiteit, Amsterdam, The Netherlands

{slblond,aoprescu,czhang}@few.vu.nl

Abstract

While diversity plays an important role in evolution and progress, it is also what makes writing Grid applications to be generally considered a challenging task. The Grid Application Toolkit (GAT) is promising to simplify this task through a plug-and-play design, which is meant to hide all diversity related application writing issues from a Grid-unaware application developer. In this paper, we report our experiences implementing and deploying a parallel MPEG encoder program as a task farming application using the GAT. While writing a real-world application following a programming model which is not Grid-specific, we identify those aspects that are made convenient by the GAT, and those that still remain cumbersome or difficult. Our main observation is that the GAT indeed simplifies application writing. However, additional services like resource brokerage are needed for similarly simplified application deployment. Though not acute, there are also some issues related to incomplete semantics.

1 Introduction

Writing Grid applications is generally considered a challenging task. The Grid Application Toolkit (GAT) is promising to simplify this task. In this paper, we report our experiences implementing and deploying a parallel MPEG encoder program as a task farming application using the GAT. We identify those aspects that are made convenient by the GAT, as compared to distributed technologies, such as RMI, and those that still remain cumbersome or difficult. Our main observation is that the GAT indeed simplifies application writing. However, additional services like resource brokerage transparent to application programmer are needed for similarly simplified application deployment. We also discuss several semantic issues of the current version of (Java-)GAT.

1.1 Paper organization

Section 2 presents shortly the real-life application we chose to implement as a Grid programming experience. It shortly introduces the reader to the history, development and critical issues of MPEG encoders. We add here a set of remarks from previous experience of implementing the MPEG encoder with RMI, and introduce GAT as a more appropriate candidate. We then continue in section 3 to a small preamble on JavaGAT, where we present the part of the JavaGAT API that has been of interest for our implementation. Section 4 details our MPEG encoder implementation as based on Joinc. A thorough description of our experience with JavaGAT is then given in section 5.

2 A Parallel MPEG Encoder

The Moving Picture Experts Group (MPEG) is a working group of ISO/IEC charged with the development of video and audio encoding standards [13]. MPEG has standardized a family of video and audio compression standards for multimedia applications [6], such as MPEG-1, MPEG-2 and MPEG-4. In fact, MPEG standardizes only the bitstream format and the decoder while giving out the liberty to encoder implementations as long as they produce bitstreams conforming to the specified bitstream format. MPEG files are much smaller for the same quality compared to other video and audio coding formats [15]. MPEG is frequently used for video/audio protocols, partly due to its ability to handle multimedia over varying bandwidth conditions [16].

As MPEG formats are being widely used, MPEG encoding however remains costly in terms of performance. The encoding process normally requires storage and computational power far beyond what traditional home computers can provide with satisfactory performance. To illustrate, in order to encode a raw DV file, a PC may require 10 GB or more disk space and may take hours to complete. There is obviously a huge gap between the end users delight and the poor performance of encoders on a single computer. This gap has led to considerable amount of research and software development for providing an effective way of doing high performance MPEG encoding. A prevalent solution to this goal is to have a parallelized MPEG encoder working over a pool of distributed multiprocessors, especially over Grid [5]. Currently, there are many different approaches in designing such a parallel MPEG encoder with either fine grain or coarse grain parallelism [8]. A typical design example is based on Master/Slave model, under which the master dispatches jobs to slaves for data processing and assembles results together in the end. Although the idea of parallel MPEG encoder is straightforward, a number of intricacies lie can be found in the design, implementation and deployment processes. Even for advanced developers it remains a very challenging task to implement a parallel MPEG encoder using distributed resources while using only common programming tools and environments.
Moreover, the trend of parallelizing computational expensive legacy software reaches far beyond the scope of parallel MPEG encoder only. We need a more common and friendly environment for parallel programming to alleviate complexity and promote ubiquity. GAT is one candidate for such an environment that provides much simplicity. Our experience in developing an MPEG4 encoder using GAT reveals that it makes parallel programming less challenging in Grid environment, and demonstrates the feasibility to simplify application of complex parallel programming techniques in developing daily user applications.

Another candidate environments we considered using is RMI. The following remarks about our RMI implementation efforts should not be seen as RMI shortcomings. RMI is a very powerful scheme for distributed programming and this has been the goal of its design team [19]. Our point is that Grid application programmers would benefit more from an abstraction layer, such as the one provided by JavaGAT. We tried to implement a customized task farming approach with RMI. The implementation design was quite simple, as explained in section 2, but the actual code was very cumbersome. We needed to explicitly write the pre-staging and post-staging of applications files, as well as to manage submission of tasks over a number of operators that are deployed outside of the application code on each working site.

Though discovery of deployed operators is possible through environment settings, the step of manually deploying each remote object that represents an operator must be explicitly dealt with. While scheduling the jobs is still a problem when using JavaGAT, the actual submission of a job and the setting of pre-staged and post-staged files for that particular job are wrapped inside API calls. This is not the case with RMI, where explicit code needs to be written in order to copy the needed files at the working sites. Also related to task submission we would have needed to devise our own call-back mechanism to keep track of the state of submitted jobs.

Another issue related to RMI surfaces from the setup of policy files on all participating clusters, as RMI requires a Security-Manager to be explicitly set up in the remote objects JVM. This would be comparable to credentials needed by the ResourceBroker of JavaGAT, though the policy files are tied to a rather unfriendly syntax. One could also choose Policy objects rather than files, but then the problems are moved in the code writing section. On the other hand, using JavaGAT and simple certificate credentials, one simple script prepares the environment for the whole application to run during a user-specified amount of time.

3 The (Java-)GAT

3.1 Context

Computing Grids are getting more and more important. While this fame is pushing people to always come up with smarter functionalities, it also creates interoperability problems. This is maybe the reason why, today, only few complex Grid application are widely used.

Nowadays, applications written for a given Grid often have to be heavily modified or even rewritten to be ported on another one. One issue here is that Grid developers have to know the intricate details of all the resource manager systems (RMS) in order to write portable applications. A second is that the same application cannot run on different RMS spanning multiple clusters even if their administrations trust one another. It implies that one cannot simply, blindly, download a Grid application programmed for a given RMS and execute it on another – it seems that Grid applications are just like desktop applications 20 years back from now.

To circumvent the situation, some projects have worked on integration facilities for their own system [14, 9], while others tried to define more general interface on top of which applications can be written independent of the underlying RMS [3]. While being an important step forward, two issues still remain. First if multiple solutions are commonly adopted, even if alleviated, the interoperability problems will persist. Additionally, none of this solution intend to simplify life of the programmers and their APIs still remain complicated to learn and use.

This is where GAT (the Grid Application Toolkit) steps in. By providing a generalized and intuitive interface, GAT will allow the developers to do not even have to know which RMS is present on the underlying Grid. Even better, GAT will choose at runtime the correct way to access resources.

In this section we will briefly present the concepts behind GAT. We will first introduce the big picture of the GAT organization, finally we will briefly discuss the subset of the JavaGAT API we used to program our MPEG encoder.

3.2 GAT Organization

A programmer does not have to know the GATs internals to write an application on top of it. However, we firmly believe knowing a bit about GATs design helps to understand its usefulness. The interested reader is invited to refer to [1] for further details.

From figure 1 we can see that the GAT layer is split into 3 logical components. The API is the only piece of GAT the programmer has to know and to deal with when writing Grid applications. Its function is bound to the strict minimum – it provides the application with calls for essential Grid operations in a simple and stable way. The GAT engine will decouple applications from the always

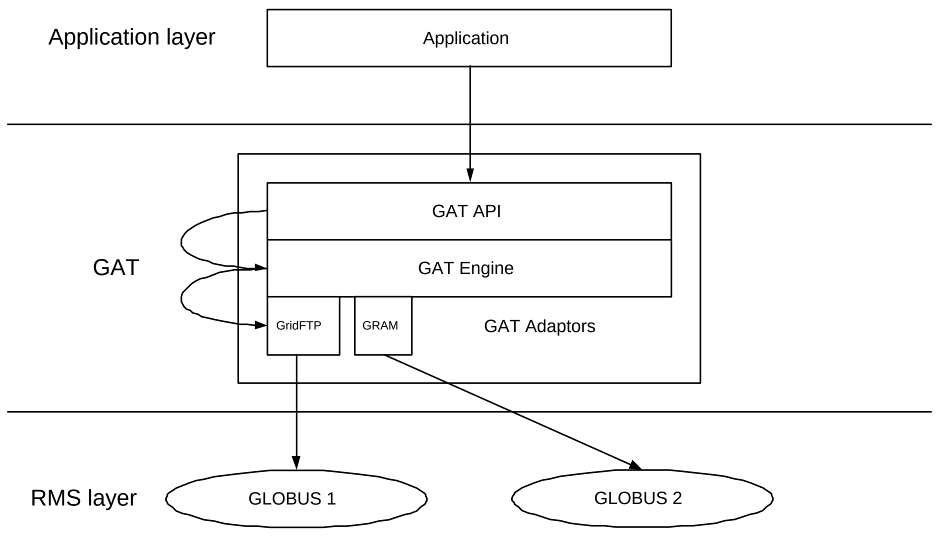


Figure 1: The GAT framework

changing middleware by loading adaptors when needed. The engine is supposed to be very thin, transparently providing an efficient way to link the logic of Grid services with the GATs API.

Finally, the engine chooses adaptors on demand in order to satisfy the capabilities defined by the API and required by the application (job submission, file transfer etc.).

3.3 JavaGAT API – The Employed Subset

3.3.1 Files

In the context of task farming, we usually have to deal with pre-staged and post-staged files. The former is the set of files the task needs access to in order to run properly. Examples of such files would be required libraries, file to be read by the task during execution, or the tasks code. Post staged files are the set of files which the task produces, and which the master application needs to retrieve after the task finished.

In order to easily deal with both pre-staged and post-staged files, we used the `File` interface from the package `org.gridlab.gat.io`. Creating an `org.gridlab.gat.io.File` object is similar to the mechanisms known from `java.io.File`, but will provide the same services for a wider variety of file access protocols. For instance, the file could now be anywhere, as long as the location can be described by an `org.gridlab.gat.URI`, and at least one of the adaptors is able to access it. The Gat Engine will make sure the appropriate adaptors have been loaded and an instance of these adaptors is representing the file.

The code listed below shows how to obtain a `org.gridlab.gat.io.File` object for a file, given its name as a `java.lang.String`. It first builds the application's context, an instance of `GATContext` (`org.gridlab.gat.GATContext`). Optionally, an instance of `org.gridlab.gat.Preferences` may be obtained, which would be responsible for specifying user preferences when selecting adaptors. If there are no specified preferences, or the `createFile` method is invoked with a `null` value for the `Preferences`, the default policy for selecting adaptors is used (note: specifying "any" in the scheme part of the Files URI will enable the Gat Engine to dynamically determine which adaptor(s) should be used).

```
String      fileName = "file";
URI         fileURI  = new URI ("any:///" + fileName);
GATContext  context  = new GATContext  ();
Preferences prefs    = new Preferences ();
File        file     = GAT.createFile  (context, prefs, fileURI);
```

3.3.2 Jobs

As described at the beginning of this subsection, for task farming we also need file retrieval mechanisms. Using the `org.gridlab.gat.resources.SoftwareDescription`, the pre-staging and post-staging may be expressed with only two lines of code, (assuming the `org.gridlab.gat.io.File preStagedFileList[]/postStagedFileList[]` have already been populated):

```
SoftwareDescription sd = new SoftwareDescription();
sd.setPreStaged  (preStagedFileList);
sd.setPostStaged (postStagedFileList);
```

If the applications tasks are designed to use a certain file as `standard input` and to redirect the `standard output` and `standard error` to some file(s), then one could use the next few lines of code:

```
sd.setStdin  (GAT.createFile (context, prefs, new URI (stdin )));
sd.setStdout (GAT.createFile (context, prefs, new URI (stdout)));
sd.setStderr (GAT.createFile (context, prefs, new URI (stderr)));
```

Finally and most important, the line of code which indicates the location of the tasks executable:

```
sd.setLocation (new URI (executableFilePath))
```

After this setup phase, the task, represented by the `SoftwareDescription` object, is ready to be submitted for execution (we refer the reader to section 5.3 for more details about job submission in JavaGAT).

4 Implementing the MPEG Encoder

4.1 GAT

As specified in section 2, we chose to implement the MPEG encoder using a task farming approach. Following the ideas provided in the previous section, we decided to use JavaGAT to make its programming and portability easier.

Inspired by the generalization of task farming applications like SETI@home [10] and Boinc [2], the JavaGAT developers decided to provide a standard similar interface so that the programmer doesn't have to know about the distribution process. That layer on top of JavaGAT is named Joinc.

4.1.1 Joinc

In the Joinc programming model, task farming applications are easily expressed by using two classes, *Master* and *Task*. The *Master* class is designed to be extended by the master object of the application, hence its abstract methods will be implemented by the application. *Task* exhaustively describes each job and its dependencies so that it can be properly executed on a remote machine.

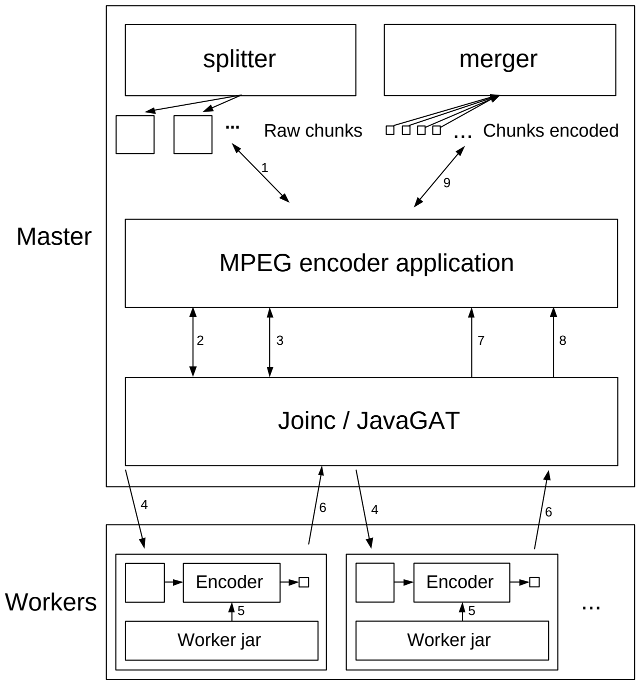


Figure 2: The MPEG encoder organization

The Master class contains a half dozen of abstract methods which will have to be implemented by the application programmer. A call to *getTask* should return a Task object with a unique task identifier, the file names of the stdin, stderr and stdout, the pre and post-staged files, the name of the class containing the main method of the task, and finally, the parameters the task accepts. A call to this method is done right before submitting a task.

totalTasks will return the number of tasks the application will produce and *maximumMachines* the number of machines Joinc can simultaneously use to run the tasks. These functions are called at the very beginning of the *start* method to initialized the behavior of the task distribution.

taskDone will notify the application of a task termination, and *idle* permits to the programmer to do what ever he wants when there is nothing better to do.

To conclude, the application calls the *start* method at the very beginning of the execution, then Joinc takes the control over the application details – the application doesn't have to deal with GAT at any moment. It is time to see how we can use this nice interface to write an application and, more precisely, a parallel MPEG encoder.

4.1.2 The Application

Using figure 2 we will now show step by step how Joinc can be used to parallelize a MPEG encoder. The remainder of this section will provide the reader with some insights about parallel MPEG encoding while illustrating how to use Joinc to write applications.

The very first thing the encoder does at step 1 is to execute the tool *avisplit* to divide the raw AVI files into smaller chunks which can be shipped to the workers. Note that *avisplit* will create independent chunks, that means that once the tasks are dispatched among the workers, they will compute without having to care about interleaving frames, or any other communication. The encoder will keep track of the generated chunks names and associate one chunk with one task.

Then the *start* method is called by the application to let Joinc take control. Joinc subsequently calls some initialization methods to learn about the encoder settings. At the end of step 2, Joinc knows the number of tasks it has to submit as well as the number of machines it is allowed to simultaneously submit tasks to.

Joinc then has to retrieve the next task object from the encoder through the *getTask* method call (step 3). This method will initialize the fields of the *Task* object, so that each worker encodes a part of the raw file. To do so, Joinc initializes a *SoftwareDescripton* per task so that GAT can transparently handle file transfers and job submission at step 4.

Step 5 corresponds to the execution of the worker code by the remote Java virtual machine. One advantage of Joinc over Boinc is its ability to send a worker jar over the network instead of having to install the binaries at all sites beforehand. Additionally to these flexibility issues it is also important to note that this constitutes a method to distribute any kind of computation in a portable way. In our example, the minimalist worker will just execute the tool *mencoder* [12], which will encode the chunk – this is the computing intensive task we want to parallelize. Once done, the now compressed movie chunk is shipped back to the master (step 6), and Joinc calls the *taskDone* method (step 7) for this task.

After all the tasks complete, Joinc will simply return control to the application (step 8). This is time to execute the tool *avimerge*, which merges all the chunks into the compressed movie (step 9).

5 The Good, The Bad And The Ugly

In this last section we will sum up what the experience of using JavaGAT taught us. While being a generally positive experience, we also encountered a number of problems or noticed a lack of middleware facilities when trying to implement more advanced features in our encoder. We first describe what the use of GAT bought us, and will list those parts we feel are general enough to be part of GAT, and we then will end with technical issues that should be improved to enhance the programming experience with GAT.

5.1 The Good

As we have seen in the examples presented in section 3.3, Grid programming becomes easier when assisted by the JavaGAT API. The details of different approaches to achieve a given operation on a given object are well hidden inside the adaptors implementing the respective object interfaces. The GAT engine will dynamically load a set of appropriate adaptors and delegate the requested operation to them. Though not required, the user is able to specify which adapter should be used for a given object, through the `preferences` passed to the GAT engine at instantiation time, as shown below.

```
Preferences prefs = new Preferences ();
preferences.put ("file.adaptor.name", "gridftp");
File file = GAT.createFile (context, prefs, fileURI);
```

There is a high degree of flexibility, without diminishing the highest achievable degree of transparency – more about the limitations on transparency can be found in section 5.3. The benefits of hiding details are best shown in the small code excerpt given below, with the use of `context` and `preferences` as discussed in section 3.3.1. The intricacies of creating an object that would communicate to a RMS are kept from the Grid developer, which only has to create an object with the `org.gridlab.gat.resources.ResourceBroker` interface, an action comparable in coding effort with the creation of a `org.gridlab.gat.io.File` object.

```
GATContext     context = new GATContext ();
Preferences    prefs   = null;
ResourceBroker broker = GAT.createResourceBroker (context, prefs);
```

Another good feature of the JavaGAT design is the availability of different levels of error messages. While for adaptor writers it is recommended to use the `gat.debug` system property when testing their programs, the rest of the users should consider the use of `gat.verbose` system property for high level debug purposes.

5.2 The Bad

To us, the functionality in GAT we missed most is its lack of brokerage awareness. Admitting the fact that Grids will get larger and larger, we think it is a mistake to let the programmer deal with the brokerage issues. The same remark applies to scheduling, where some very basic default scheduling policies, like FIFO, would often provide enough functionality for simple load balancing.

At the moment, the only way for the GAT user to have its application automatically scheduled on multiple clusters is to use the GRMS adaptor [18]. While GRMS provides an optimal transparency to the programmer, its not fully stable yet and still under development. The cost for this transparency will then come with poor performances to schedule jobs over an entire Grid – the issue is to know if using GRMS does not cost more than it buys by taking smart scheduling decisions.

We believe a tradeoff could lie in a distributed flavor of GRMS, possibly making decisions based on partial knowledge. We could for instance imagine to have a simple brokerage adaptor running at each GAT application and take decisions based on samples received from an external services like [11] – please note that this scheme would be quite similar to the 'Smart Sources' mechanism used in peer-to-peer systems to prevent the 'best' nodes to be flooded with requests when using a centralized information system component.

Finally, making GAT aware of its environment would improve its transparency, e.g. by letting it negotiate with clusters and determine the resource management system they are running. We saw in section 3 that we have to specify in advance which cluster is using which system. This substantially complicates the programming when having to submit jobs over heterogeneous resource management systems. Having GAT performing service discovery would, once again, help the programmer to write portable Grid applications.

5.3 The Ugly

As promised in section 5.1, we will try to list some of the limitations of the provided transparency we stumbled upon while developing our MPEG encoder application. One issue is related to credentials: As the application is Grid-empowered, the need of valid credentials is critical. Still, validating the credentials is bothersome action and automating the process would help. We tried to figure out where the problem actually lies, and it turns out it is not the responsibility of JavaGAT, nor of Globus credentials management system. In this case, the achievable degree of transparency is dependent on the Certificate Authority rules for certificate protection. In our case, that CA is run by DutchGrid [4], and does not allow a certificates pass-phrase to be empty. This is not the case with SSH authorization system, which is based on private/public key protocol and which allows for pass-phrases to be empty, hence allowing the login process to be fully automated. As a conclusion, the limitation on transparency where credentials are involved is strongly related to credentials specific protection mechanisms, which are under the strict control of the security component governing the given Grid.

Another issue is related to the main topic of section 5.2: In the current state of JavaGAT where ResourceBroker adaptors are concerned, the Grid developer which would like to have the application run on the real Grid and not on the local machine, needs to use the following code excerpt, which may not be very intuitive:

```
GATContext      context = new GATContext  ();
Preferences     prefs   = new Preferences ();
preferences.put ("ResourceBroker.adaptor.name", "globus");
preferences.put ("ResourceBroker.jobmanager",   "pbs");
ResourceBroker  broker  = GAT.createResourceBroker (context, prefs);
```

While it is true that the overload of dynamically deciding whether the application should run on the Grid or on the local machine is not worth the transparency gain, a compromise could still be made. The user should be able to indicate that the application needs to be gridified in a more abstract fashion, e.g. by using:

```
GATContext      context = new GATContext  ();
Preferences     prefs   = new Preferences ();
preferences.put ("ResourceBroker.type", "remote");
ResourceBroker  broker  = GAT.createResourceBroker (context, prefs);
```

In JavaGAT, to create the incarnation of a Joinc task (`Job`), the user has not only to provide the description of the software representing the task, but also a description of possible run-time environments for this task. This is achieved by specifying a `org.gridlab.gat.resources.ResourceDescription` (a set of software and hardware run-time requirements), or a `org.gridlab.gat.resources.Resource` (representing a specific resource). Once the `JobDescription` object has been created it can be used to obtain as many `Job`s for the task as the user would like to have. The `Job` object is returned by the `broker` after successfully submitting a job that meets the `JobDescription` specifications.

```
Map attribs = new Map ();
attribs.put ("machine.node", "fs0.das2.cs.vu.nl");
ResourceDescription rd = new HardwareResourceDescription (attribs);
Job j = broker.submitJob (new JobDescription (sd, rd));
```

Finally, a rather technical issue was raised by the `gridlab.gat.resources.Job.getJobID` method. Though the return value is a *"globally unique identifier for the physical job corresponding to this instance"* [7], it is no longer available after the `Job` object state changed to something different than `Running` or `Submitted`. When many jobs need to be monitored, as is the case of our MPEG encoder implementation, the globally unique identifier might prove a better way to select finished jobs from a list of all submitted jobs, rather than using the jobs object reference. Again, this is a technical issue, but following the definition of a globally unique identifier as given in [17], one might wonder why the job IDs cannot be retrieved from the job object in *any* given state of the job.

6 Conclusion and future work

In this contribution we discussed our experience writing a Grid-parallel MPEG encoder using the Java version of GAT. After having briefly introduced what an MPEG encoder is and in which context it can be distributed, we talked about the general issues GAT is supposed to solve, and further explained how GATs design team chose to address these issues. We finally came to the main point of this paper by showing to what extend GAT simplifies application development.

We then presented the approach we chose to parallelize and implement the MPEG encoder. By introducing an additional layer between the application and GAT we showed that it is possible to completely hide the distribution from the user, while at the same time making Grid programming portable and straightforward. As an example we gave some insights about a previous experience we had with using RMI for parallelizing the same application. However, distributed programming schemes have their own applicability and our point is not that RMI should be generally replaced by GAT, but merely that Grid programming without Grid-awareness is better served by GAT.

The overall conclusion after working with JavaGAT is optimistic. Though there are bad and ugly things to be considered, neither of them seems to be insurmountable. In our opinion, it is just a matter of further development and real-world user feed-back to make the JavaGAT a truly transparent and flexible Grid application development toolkit.

We are now planning to port our MPEG encoder application on the worlds first "virtual city supercomputer", the *Almere Grid* testbed in the Netherlands. While still only few information are available about the project, there would already be more than 2.000 machines connected through the 100 Mbit/s fiber network built across the city. This would be an incomparable environment to introduce one of the very first Grid application useful for users.

7 Acknowledgments

First, we would like to thank Prof. Henri Bal for having taught us parallel programming and the suggestion to parallelize a real world application – a MPEG encoder. We also want to express our gratitude to Jason Maassen for having been an impassioned Grid computing assistant through our first masters year and the initiator of Joinc. Special thanks go to Rob van Nieuwpoort for being the "back office" guy of JavaGAT. Finally, this paper would have never been written without the long lasting support of Thilo Kielmann and Andre Merzky.

References

[1] G. Allen, K. Davis, T. Goodale, A. Hutanu, H. Kaiser, T. Kielmann, A. Merzky, R. van Nieuwpoort, A. Reinefeld, F. Schintke, T. Schütt, E. Seidel, and B. Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[2] D. P. Anderson. BOINC: A System for Public-Resource Computing and Storage. In *Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, 2004.

[3] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Lecture Notes in Computer Science*, volume 1459, page 62, Jan 1998.

[4] `http://www.dutchgrid.nl`.

[5] I. Foster. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 2nd edition edition, 2004.

[6] D. L. Gall. MPEG: A Video Compression Standard for Multimedia Applications. *Communications of the ACM*, 34(4), 1991.

[7] `http://www.cs.vu.nl/~robn/gatdocs/`.

[8] S. Y. I. Assayad, V. Bertin. Parallel Model Analysis and Implementation for MPEG-4 Encoder. In *Proceedings of Embedded Processors for Multimedia and Communications II*, San Jose, CA, USA, 2005.

[9] D. B. Jackson. *Grid Scheduling with Maui/Silver*, chapter 11. Kluwer, 2003.

[10] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, and M. Leboisky. SETI@home-massively distributed computing for SETI. In *Computing in Science & Engineering* `[see also IEEE Computational Science and Engineering]`, volume 3, Jan 2001.

[11] J. Maassen, R. V. van Nieuwpoort, T. Kielmann, and K. Verstoep. Middleware Adaptation with the Delphoi Service. In *Workshop on Adaptive Grid Middleware (AGridM 2004)*, Juan-les-Pins, France, 2004.

[12] `http://www.mplayerhq.hu/homepage/design7/news.html`.

[13] `http://en.wikipedia.org/wiki/MPEG`.

[14] B. Nitzberg, J. M. Schopf, and J. P. Jones. *PBS Pro: Grid Computing and Scheduling Attributes*, chapter 13. Kluwer, 2003.

[15] R. Pulles and P. Sasno. A set top box combining MHP and MPEG-4 interactivity. In *Proceedings of the 2nd European Union symposium on Ambient intelligence*, Eindhoven, Netherlands, 2004.

[16] Rob Koenen, ed. Overview of the MPEG-4 Standard, March 2002. [Online]. Available: `http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm`.

[17] A. Tanenbaum and M. van Steen. *Distributed Systems, Principles and Paradigms.* Prentice Hall, 2002.

[18] The GridLab Project. (2003) The GridLab Resource Management System. [Online]. Available: `http://www.gridlab.org/grms/`.

[19] J. Waldo, G. Wyant, A. Wollrath, and S. Kendall. A Note on Distributed Computing, Nov 1994.

Implementation of Fault-Tolerant GridRPC Applications

Yusuke Tanimura, Tsutomu Ikegami, Hidemoto Nakada, Yoshio Tanaka, and Satoshi Sekiguchi
National Institute of Advanced Industrial Science and Technology
Grid Technology Research Center, 1-1-1 Unezono, Tsukuba Central 2
{yusuke.tanimura, t.ikegami, hide-nakada, yoshio.tanaka, s.sekiguchi}@aist.go.jp

Abstract

In this paper, a task parallel application is implemented with Ninf-G which is a GridRPC system, and experimented on, using the Grid testbed in Asia Pacific, for three months. The application is programmed to run for a long time and typical fault patterns were gathered through tens of long executions. As a result, unstable network throughput was determined to be one of the biggest reasons for faults. Then, an important point for application developers is stressed, reminding them to avoid serious decline of task throughput during operations for faults, by timeout minimization for fault detection, background recovery and duplicate task assignments. This study also issues a steer for design of the automated fault-tolerant mechanism in a higher layer of the GridRPC framework.

1. Introduction

Recently, much has been expected of the Grid computing infrastructure in terms of large-scale computation for scientific discovery. A user will always have reasons to use a remote computer, but the Grid offers hope for scientists to be able to finish several-years-computation in several weeks. In addition, the Grid middleware technology that the Globus Toolkit[1] represents is being developed and matured based on past research. Indeed, various experiences on development and evaluation of Grid-enabled applications, such as a climate simulation[2], solving Einstein's equations[3], and etc., have been reported and they showed the possibility of the Grid as a ready and realistic infrastructure to be used by real science. There is, however, still a severe problem caused by unreliability of the Grid towards making the Grid a infrastructure for large-scale scientific applications.

Sudden faults and scheduled/unscheduled maintenance of networks, machines and software interrupt the promise of long running applications. For example, it is usual that a hard disk on a computing node in a cluster composed of more than 100 nodes will crash at least once in several months. A Grid-enabled application should be fault-tolerant and it is expected to have capabilities to detect faults appropriately and cope with the faults without terminating the entire execution of the application. In order to implement fault-tolerant applications, there is a need to develop applications and advanced middleware to overcome the instability of the Grid, but few discussions about what kinds of faults will happen and how middleware and applications can cope with those faults.

Several researches on fault-tolerant mechanisms have been reported, however the proposed scheme could be applied to a specific category of applications or immature for use by real applications. For example, Condor[4] implements checkpointing and process migration, however they are practically usable

Table 1. Major APIs and the corresponding error codes

API	Possible fault detected	Error code
grpc_function_handle_init()	DNS query failure	GRPC_SERVER_NOT_FOUND
	Server machine is down.	GRPC_SERVER_NOT_FOUND
	Network to server is down.	GRPC_SERVER_NOT_FOUND
	Globus-gatekeeper is not running.	GRPC_SERVER_NOT_FOUND
	GRAM invocation fails at authentication.	GRPC_OTHER_ERROR_CODE
grpc_function_handle_destruct()	–	–
grpc_call()	TCP disconnection during data transfer	GRPC_COMMUNICATION_FAILED
	RPC failure to disconnected server	GRPC_OTHER_ERROR_CODE
grpc_call_async()	TCP disconnection during data transfer	GRPC_COMMUNICATION_FAILED
	RPC failure to disconnected server	GRPC_OTHER_ERROR_CODE
grpc_cancel()	–	–
grpc_wait_any()	TCP disconnection (for nonblocking data transfer)	GRPC_SESSION_FAILED
	Hearbeat has timed out.	GRPC_SESSION_FAILED

only to a single job submission. Several works on fault-tolerant MPI have been reported[5, 6], however they are still in the research phase and still immature for use by real applications.

In this research, we investigated how to make Grid-enabled applications fault-tolerant. We have developed a task-parallel application with Ninf-G which is a GridPRC system[7], and made experiments on, using the Grid testbed in Asia Pacific, for three months. Our application implements simple fault-tolerant mechanism in order to run as long as possible. Through a long-term experiment, we analyzed fault patterns that actually happened and improve fault-tolerant functionality in the cost of fault detection and recovery operation, towards future support by the higher library of the GridRPC. In this paper, we summarize several notes in developing an application that needs a long run, and also indicate what functions middleware should offer. In Section 2, we will give an overview of the GridRPC and the Ninf-G library. Section 3 describes the implementation of a fault-tolerant application with Ninf-G. Details of the long-term experiment, experimental results and insight gained through the experiment are described in Section 4. Section 5 describes related works and Section 6 gives summary and future works.

2. Ninf-G design for fault-tolerance

Ninf-G is a reference implementation of the GridRPC. In this section, the basic concept of developing the GridRPC application, which has the fault-tolerance, using the Ninf-G, is introduced.

2.1 Overview of GridRPC

GridRPC[8], which is an RPC mechanism tailored for the Grid, is an attractive programming model for Grid computing. The programming model is that of standard RPC plus asynchronous, coarse-grained parallel tasking, in practice there are a variety of features that will largely hide the dynamic, insecure, and unstable aspects of the Grid from programmers. The GridRPC API has been proposed for standardization at the GGF (Global Grid Forum, http://www.gridforum.org) since 2003, in order to make the compatibility between several RPC-functional Grid systems.

Ninf-G[7], which is currently Version 2 and built on the Globus Toolkit Version 2, provide the GridRPC-based programming model to develop task-parallel applications easily. This work is performed with Version 2.2.0 of Ninf-G released on September 10, 2004. So far, a climate simulation[2] and a replica exchange Monte-Calro simulation[9] have been implemented to evaluate the basic performance and scalability of Ninf-G. However, there has been no deep analysis of fault-tolerant application development.

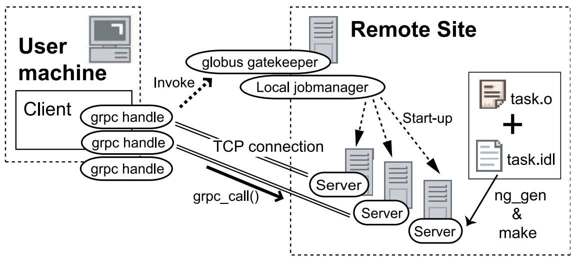


Figure 1. Connection between client and server of Ninf-G

Basically, GridRPC is a client-server model and it is assumed that some server processes are running on a remote resource to execute a specific routine. In developing a task-parallel application, both client and server programs are implemented separately and the client program calls the GridRPC function to carry out the remote routine. This paper describes ”server” as a Ninf-G server process and ”client” as a Ninf-G client process. Those terms don’t indicate a specific machine or computer resource.

Regarding the GridRPC programming and the Ninf-G behavior, a client creates several function handles each of which represents a mapping from a function name to an instance of that function on a particular server. A TCP connection between client and that server is established at this moment. Afterwards, all RPC calls using that function handle will be executed on the server specified in that binding. In the case of asynchronous RPC, the client can wait for a specific RPC using the Session ID given in the asynchronous call function. The Session ID is an identifier representing a particular asynchronous call and it can be utilized to cancel the call or check the status of the RPC. At the end of the program, all handles are released to end each server with the TCP disconnection. The client keeps those TCP connections during this flow, such as that shown in Figure 1. The connection must be established and maintained normally so that the application works. In other words, any faults will be detected on the client by looking at the status of all connections to the servers.

2.2 Fault-tolerance and GridRPC API

For application drivers, scheduled/unscheduled interruption for networks and machines cannot be avoided on the Grid environment when running their applications. An application program should continue to be processed in spite of that interruption, or it could be restarted midway. Although the middleware is expected to support applications that can do those things without complex operations required of the users, it has not been available at the production level, as described in Section 1. On the other hand, the GridRPC standard provides a primitive API set, and fault-tolerant functions should be supported in higher APIs. According to the GridRPC design, the primitive API detects any faults appropriately, and informs the application program rapidly. Specifically, the called GridRPC API should return a corresponding error code. In the case of that the error code will not be back, the heartbeat function provided by Ninf-G will be useful for an application program to avoid the deadlock.

2.3 Error handling

Table 1 shows major GridRPC APIs and the error codes that will be returned in encountering errors for the Ninf-G client. Discussion of the error codes is on-going at one of the working groups of GGF, but Ninf-G provides them based on the draft. As Table 1 shows, different kinds of faults correspond to the same error code, and it is impossible to know what happened or why the fault happened just by reading the error codes. Because the error codes only show the status of the system, application developers can describe how their program should work, based on that status. For example, GRPC_COMMUNICATION_FAILED means there is a disconnection between client and server on the RPC route. After the error occurred, the handle is invalid until the client recreates the handle as it is programmed. The application developer of Ninf-G should describe in the client program when the client restarts the server, or where it does so.

2.4 Heartbeat function

Ninf-G provides a heartbeat function that sends a small packet periodically from server to client, in order to avoid disconnection by firewall software because there is no packet communication for a certain period of time. This function is also utilized to check if the server works without errors. If many packets are dropped on the network, this function will prevent the client from waiting for incoming data and being blocked forever.

An application user should set a timeout value to close the TCP connection if no packets arrive within the timeout seconds. If timeout has occurred, grpc_wait_any() would be returned with the GRPC_SESSION_FAILED code and the server handle would be invalid.

3. Implementation of TDDFT with fault-tolerance

A sample program reviewed in this study implements a TDDFT (Time-Dependent Density Functional Theory) equation[10]. TDDFT is a molecular simulation method in computational quantum chemistry, and it treats multiple electrons excitation directly, using techniques from quantum mechanics. An original program of TDDFT contains a hotspot to be processed in parallel using hundreds of CPUs, and the hotspot is iterated thousands of times to get an accurate result. This iteration requires a long-time execution. In our implementation, therefore, the hotspot is divided into several tasks that are executed in parallel. Figure 2 shows the simple flow of our TDDFT calculation. The flow consists of 1) Configuration of input data and parameters, 2) Multiple-tasks execution in parallel, using Ninf-G and 3) Calculation of the sequential part. 1) is only performed in the first loop and then 2) and 3) are repeated as times as a user wants.

In the Ninf-G application, a user should have responsibility for creating a server handle which is related with the server that the user wants to use. This insists on user-level implementation to manage the servers to which a task is assigned and to handle down servers for fault tolerance. For the long run of TDDFT, simple server management and task assignment methods are implemented including fault discovery and recovery operation, which are explained in this section.

3.1 Ninf-G server management and task assignment

As shown in Figure 2, each task of 2) must be processed after 1), which includes reading input data about molecules and setting given parameters. In this paper, the first RPC is named the ”initialization method,” and the second RPC is named the ”main method.” An array generated by the initialization method is maintained and can be referenced by the main method; that is a feature of the remote object provided since Ninf-G version 2. In particular, TDDFT requires the same initialization on all servers, before performing the main method. The simple flow of a TDDFT process is to initialize an array on all servers and then submit one task to them. A server that has finished the task will be given another task

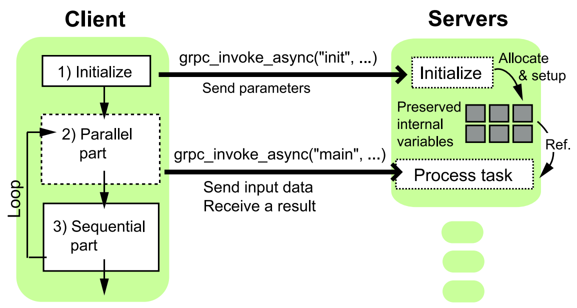


Figure 2. Implementation of TDDFT using Ninf-G

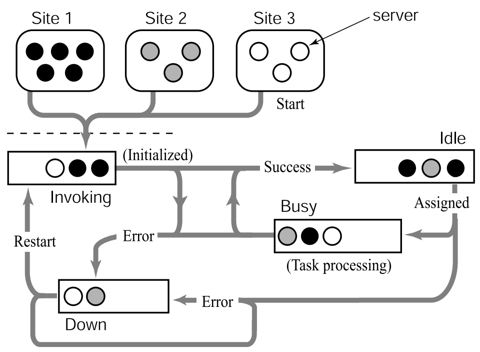


Figure 3. Status graph of the server

by the client. All tasks in each time step must be complete before the next step. On the basis of the flow, our program implements status management of all Ninf-G server processes as shown in Figure 3.

a) The status of a spawned server process is ”Invoking.”
b) The status moves from ”Invoking” to ”Idle” after completion of the initialization method.
c) A task is assigned to the ”Idle” server and its status is ”Busy” during calculation.
d) The status returns to ”Idle” after a task has finished without error.

3.2 Fault discovery and the recovery operation

In this paper, our motivation to implement fault tolerance is to enable a long-time execution on the real Grid. To achieve this goal, the client should not stop at any faults and the server which suffered the faults should be restarted when it becomes available again. Those two points are least requirements for our experiment and they are implemented in the application program.

When faults happen, the TCP connection between client and server is closed and the server process automatically exits. In the client, an appropriate error code is returned by the GridRPC system as described in Section 2.3. Therefore the client is programmed to change the status of the server which returns the error to ”Down.” Any tasks will not be assigned to ”Down” server any more.

For the server process to be restarted by the client, the client is programmed to destroy the server handle and create it after the fault is solved. Subsequently, the initialization method is called because the remote-object function is utilized in this case. When those processes are completed, the status of the server moves to ”Idle.” Since this TDDFT program requires the same initialization method on all servers and the calling parameters are static, it is possible to register the initialization method so that it is called again at recovery time.

A recovery operation is implemented to check the status of all servers every hour, find a handle array whose handles are all invalid, and recreate the handle array. The handle array is a set of handles on compute nodes which belong to the same cluster. The design of the handle array is based on the fact that the Ninf-G client requests the server process using GRAM (Grid Resource Allocation Manager) of the Globus Toolkit, and that the Globus gatekeeper also submits the server invocation to a local job manager, such as PBS or LSF. Because GSI (Grid Security Infrastructure) authentication of GRAM takes a certain overhead, the client will ask for multiple-servers invocation with one request with GRAM. Several nodes that are running a server managed by the same GRAM request will not be released even if one of them is down. The batch system is not suitable to restart only some Ninf-G servers among the servers managed by one server handle array, because normal servers will also be restarted in vain. The recovery interval should be considered when figuring the cost of this operation.

4. Experiments

4.1. PRAGMA routine-basis experiment and its testbed

A developed TDDFT application was experimented as a PRAGMA routine-basis experiment. The Pacific Rim Application and Grid Middleware Assembly (PRAGMA)[11] is an institution-based organization, consisting of twenty-seven institutions around the Pacific Rim who are dedicated to building sustained collaborations and to advancing the use of Grid technologies in applications among a community of investigators working with leading institutions around the Pacific Rim. Applications are the focus of PRAGMA and are used to bring together the key infrastructure and middleware necessary to advance application goals.

Asia Pacific Partnership for Grid Computing (ApGrid)[12] is an open community encouraging collaboration. As of the end of May 2005, 49 organizations from 15 economic were participating in ApGrid. PRAGMA and ApGrid have been collaboratively building Grid testbed in Asia Pacific region.

PRAGMA routine-basis experiment was initiated in May 2004 and its purpose is to learn requirements and issues for testbed operations and developments of Grid-enabled applications through exercise with long-running sample applications on an international Grid testbed. PRAGMA routine-basis experiment was contacted by San Diego Supercomputer Center (SDSC) and TDDFT application is one of the applications used in the routine-basis experiment. One of our interests in this experiment is to watch actual fault patterns and discuss how to implement more appropriate fault-tolerance than tentative implementation. The experiments continued for 3 months, from June 1 to August 31 in 2004. For the routine-basis experiment, we provided a dedicated testbed which includes computational resources from AIST (Advanced Industrial Science and Technology, Japan), SDSC (San Diego Supercomputer Center, USA), KISTI (Korea Institute of Science and Technology Information), KU (Kasetsart University, Thailand), NCHC (National Center for High-performance computing, Taiwan), USM (University Sains Malaysia), TITECH (Tokyo Institute of Technology, Japan), NCSA (National Center for Supercomputing Applications, USA), UNAM (Universidad Nacional Autónoma México) and BII (Bioinformatics Institute, Singapore).

4.2 Results of the long run

The client program achieved a total of 906 hours (about 38 days) execution time during the experiment. The maximum number of sites that ran the Ninf-G server at the same time was 7 and the maximum number of CPUs was 67. The longest execution started at 20:00 JST on July 7, 2004 and continued until 16:00 JST on August 4, for a total of 164 hours (about 7 days). The execution environment of the longest run used 59 servers spread over 5 sites, as shown in Table 2. The Ninf-G client ran on the AIST node and it started the servers on remote sites using the server handle array. Throughput in Table 2 was measured by sending a 1 MB message from the AIST node to the remote-site node using TCP before the experiment. This TDDFT execution was able to divide the parallel part into 122 tasks in each loop. 4.87 MB of data was transferred from the client to the server on each RPC and 3.25 MB of data was transferred from the server to the client, as the result of the calculation.

Figure 4 shows the history of the live servers without the NCHC site. The reasons that the client recognized the server as ”Down” are as follows: 1) Data transfer failed at low network throughput, 2)

Table 2. Ninf-G server environment at the longest execution

Site	#CPU	System	Throughput
AIST	28	PentiumIII 1.4GHz	116 MB/s
SDSC	12	Xeon 2.4GHz	0.044 MB/s
KISTI	16	Pentium4 1.7GHz	0.28 MB/s
KU	2	Athlon 1GHz	0.050 MB/s
NCHC	1	Athlon 1.67GHz	0.23 MB/s

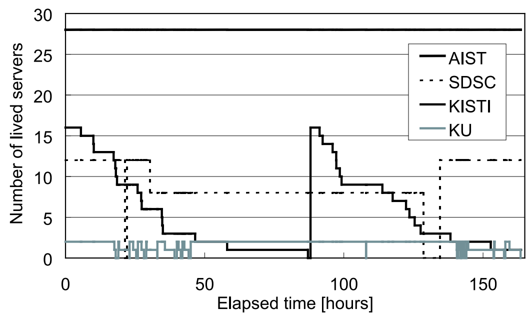


Figure 4. Transition of live servers

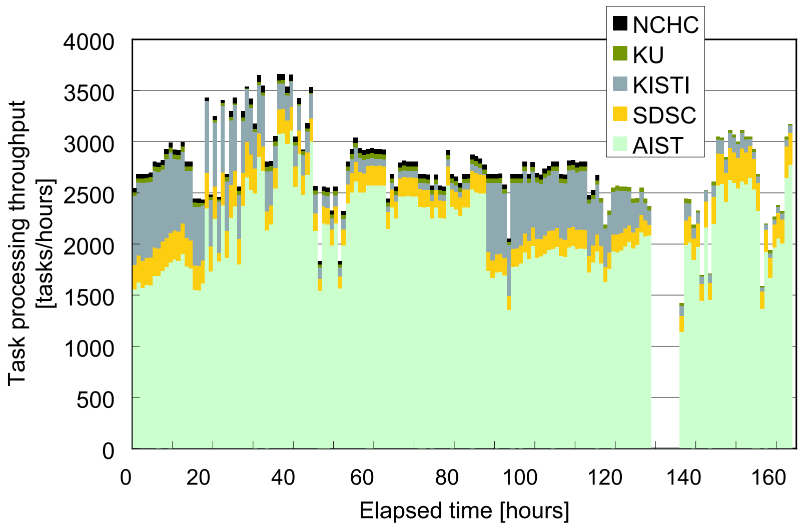


Figure 5. Transition of task throughput

The TCP connection was closed without notification to the client, and 3) The node was down because of heat problem or disk error. 1) and 3) were detected within short time by the TCP disconnection but 2) was only detected by the heartbeat timeout. Reasons identified as ”Down” servers were 67% for 1), 31% for 2), and 2% for 3) during the longest run.

Figure 5 shows the task throughput per hour. The reason that the task throughput was zero at 130 hours after start is that a boot request for the servers was queued for about 5 hours. As the execution time of one task was short, most tasks were processed on the AIST resource. If one of the remote sites was unavailable due to a fault, the task throughput tends to be higher. Nevertheless, the task throughput decreased during the time that the number of live servers was changing. Two reasons are considered as reasons for this problem. One is that the client program waits for all of the unfinished tasks each loop to be completed, and the resubmission caused by the fault might be the bottleneck. In addition, fault detection with the heartbeat takes from 300 to 400 seconds. The other reason is that the recovery operation for the ”Down” server is sequentially executed while task submission is pending. The solution is described in Sections 4.3, 4.4, and 4.5.

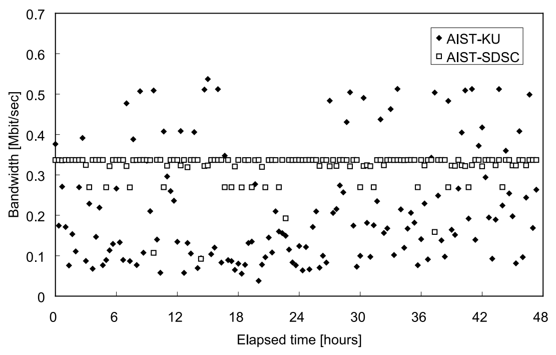


Figure 6. Transition of network throughput

Table 3. Heartbeat receiving interval and data transmission cost [sec]

	Heartbeat interval		Data transfer time	
Site	Ave.	Max.	Ave.	Max.
AIST	30.3	60.0	0.396	0.476
SDSC	30.6	81.0	19.4	25.7
KU	65.1	203	96.7	159

4.3 Stability of the network on the testbed

The KU node encountered faults more frequently than the SDSC node even when network throughput did not differ very much. Therefore, we investigated the details and the reason for the TCP disconnection. Figure 6 shows the network throughput of the AIST-SDSC path and the AIST-KU path every 20 minutes. The throughput of the AIST-KU path changed dynamically, and was sometimes very poor.

During 20 hours of TDDFT execution with 1 client on AIST, 4 servers on AIST and 4 servers on SDSC, the frequency of TCP retransmission from the client to the servers was 0.014 %. The same measurement with 1 client on AIST, 4 servers on AIST and 4 servers on KU showed 0.11 % as the retransmission ratio. The 8 times higher value also indicates that the TCP disconnection between AIST and KU was caused by an unstable network.

4.4 Improvement of fault detection

When the TCP socket was closed normally by a data transfer error of data transfer, etc., Ninf-G can detect the fault instantly. However, the detection will be delayed by at least the timeout seconds when the fault can only be found by the heartbeat timeout. But, if a user sets too small a number as the timeout value, a small delay will be judged as a fault and a normal connection may be terminated. We experimented how many seconds the heartbeat might be delayed by, in order to set an appropriate heartbeat timeout value.

Table 3 shows the receive intervals of the heartbeat with the program execution on 4 servers each on AIST, SDSC and KU. The servers on AIST and SDSC send a heartbeat packet every 60 seconds. The servers on KU send it every 80 seconds. If the client has not received any data or heartbeat by the time the interval has passed 5 times, that is, no data for 300 or 400 seconds, the RPC operation will be judged to be a heartbeat error. However, the heartbeat uses the same connection as the data transmission, so the heartbeat will not be received during data transmission. Instead of that, data arrival will be treated as a heartbeat reception. For this reason, the appropriate timeout value depends on how many seconds data transmission takes on average, and at the maximum, such as shown in Table 3.

In our experiment, the heartbeat, from SDSC and KU were delayed but the longest heartbeat arrival interval was shorter than the longest data transmission plus the heartbeat sending interval. Therefore, the total timeout seconds to give-up the RPC seems better to be set slightly longer than the longest data transmission, and shorter than the longest data transmission plus the heartbeat sending interval. On the other hand, it is effective that the heartbeat sending interval is shorter, and the allowed count of the

Table 4. Cost of the recovery operation [sec]

	AIST	SDSC	KU
1) Server halt	0.00709	0.00465	0.854
2) Server check	0.556	2.37	1.67
3) Server restart	4.83	10.6	2.80
4) Initialization	6.74	3.67	48.1
Total time	12.1	16.6	53.5

Table 5. Cost of the background recovery [sec]

No fault	215.62
Background recovery	216.26
Foreground recovery	263.35

heartbeat sending interval passed is large, in order to save on the cost of timeout wait.

4.5 Improvement of recovery operation

4.5.1 Recovery cost

From the results in Figure 5, it is revealed that the recovery cost affects task throughput. It is important to estimate the recovery cost appropriately and to minimize the cost by adjusting the timing or frequency of the operation. The cost of each recovery procedure was measured on 3 sites, AIST, SDSC, and KU and summarized in Table 4.

First of all, it is necessary to restart the server by releasing the server handle. In the case of starting the servers using the handle array, even not ”Down” servers are also restarted. The release of the handles to halt the server is shown in 1) in Table 4. 2) is used to request the globus gatekeeper to start the servers again. The client tries to get access to the globus gatekeeper, pass authentication, and request to the local scheduler. After all of these tasks are done, the server starts normally and notifies this to the client in 3). 4) is an operation used to call the initialization method to the server after 3). The cost of 4) depends on the particular application. The total cost includes operations 1) to 4).

The results shown in Table 4 represent the best value obtained out of 3 trials. The difference among 3 sites comes from divergence of processor speed, network performance, configuration of the local job manager and Globus, and etc. More cost to restart the KU servers might be required, depending on the network throughput. 3) and 4) take more time than 1) and 2) but those are server-side operations. The client should be processing other things while 3) and 4) are going on so that the costs of 3) and 4) are hidden from the client. If the recovery failed due to continuous network problems, etc., an error will occur in 2). 2) can be summed up as many retries of the recovery, but it can also be hidden by background operations with thread programming.

4.5.2 Background recovery

On the basis of the discussion in the last section, the background recovery was implemented with one thread that performs operations 1) to 4). For evaluation purposes, 2 sets of 6 servers were started on the AIST node to avoid unstable network performance and heterogeneity of computer performance. One of the sets was halted in the first loop during task processing, and the recovery operation was done after the loop. The foreground recovery process sequentially performs 4) for each server and waits for the recovery before the sequential part of TDDFT. The background recovery is moving on to the sequential part of TDDFT while the recovery is being tried. Table 5 shows how many seconds the calculation took for the sum of the first loop and the next. It was found that the background recovery took almost the same time as execution without the halt.

4.6 Discussion

In this fault-tolerant application development for a long run, the fault detection is implemented by catching the error code of the GridRPC API. Then, subsequent process is cancelled, a failed job is resubmitted to another server, and an affected server is recognized as ”Down” on the error condition. Nevertheless, it is not possible to detect the situation that a client freezes by waiting for unreceived packets for all time. The heartbeat function is necessarily enabled in execution to exit the trouble. In the experiment on the ApGrid/PRAGMA testbed, most faults were caused by decline of the network throughput. We learned the following points from our long-run experiment in order to make a sophisticated fault-tolerance mechanism into the development and execution of the GridRPC based application:

- The heartbeat function is useful to detect the fault on an unstable network. To improve usability of this function, it is expected to configure the timeout value automatically because the best value is not easy to be determined statically and easily.
- In case that the initialization method takes time, the cost of the server recovery will be large and processed in background. This issue is obvious but users should care more when a task requires initialization. In addition, the cost of the initialization method depends on the application.
- There is no method to determine how long the fault might continue. Periodic trials of the recovery cost a certain time, but it is revealed that the cost will not be large as a proportion of the total time. Naturally, a background trial is expected in implementation of the higher-level of GridRPC API.
- There is an issue with how some ”Down” servers can be treated by the same handle arrays as other normal servers, and can be restarted. A tradeoff exists between the restart cost of normal servers in vain, and no use of the servers that can be used at this restart. This should involve some functions of the local scheduler.
- When the remaining tasks are few, duplicate task assignment is very effective because failure of the final task increases waiting time due to the need for recalculation. Our results showed serious decline of task throughput without duplicate assignment.

5. Related work

There are other middlewares such as Condor MW[13] and Ninf-C[14] based on Condor, that implement a similar function to RPC. They achieve the checkpoint and the task migration at the middleware level, and an on-going-task can be restarted from some midpoint on another server. Netsolve[15], another implementation of GridRPC, provides automatic resubmission of a failed task to another server but no restart from a midpoint. In contrast to those middleware suites, Ninf-G does not provide a specialized information server that can treat a Ninf-G task. Ninf-G is designed such that fault-tolerance and server management should be implemented in the application level or in the higher middleware. Our work is important from the viewpoint of discussing fault-tolerant application development using the primitive GridRPC API. This will bring higher functionality of the GridRPC.

The faults we met in this experiment depend on the current Ninf-G that attempts to keep the TCP connection between client and server. The reason for keeping the connection is to obtain better performance for fine-grained task parallel processing. This paper does not focus on the issue of the long establishment of a TCP connection. NetSolve, which does not keep the connection, will not have an effect on the client program so much as against a network fault. However, the discussion about the failure of the heartbeat or data transmission is still necessary. The lessons learned from our experiment are useful for long-run applications to save costs in fault detection and server recovery.

6. Conclusion and future works

In this paper, several important points to be considered in the development and execution of a task-parallel application that can be run for a long time on the Grid have been discussed. Our client program was simply implemented with consideration to the error handling of the GridRPC for fault detection and management of the Ninf-G server to assign a task or to recover down servers. The recovery operation supported the remote object, too. Through the routine-basis experiment, typical fault patterns and ratio were inspected. Then our tentative implementation was improved for efficient resource utilization. In particular, minimizing of timeout-based detection, background recovery, and duplicate task assignment in advance were rewarding. However, they are plainly complicated in implementation and could be reproduced by several application users. They should be implemented in middleware and the result in this paper is a signpost to the design of a higher-level interface of the GridRPC and an increase in its functionality. In the new interface, we plan to design an easier function to submit multiple tasks one time over the primitive GridRPC APIs, including automatic and quick server recovery.

In addition, this paper raises an issue of faults at remote side, while the client is a single point of failure and it is assumed that users choose a reliable machine to run their clients. As our future works, restart of the client, rollback and resubmission of the task which was interrupted by client's fault, should be designed in the higher layer over the GridRPC.

Acknowledgement

We would like to thank to Dr. Nobusada (Institute of Molecular Science) and Dr. Yabana (Tsukuba University) who provided their TDDFT program for our research work. Also, we would like to give thanks to Mr. Takemiya (AIST) who gave us many comments on our work.

This work has been conducted as a PRAGMA routine-basis experiment and supported by PRAGMA and Ap-Grid communities. We are grateful for their resource contribution to our long term experiment. We would especially like to give thanks to Ms. Cindy Zheng (SDSC) who gave great leadership for conducting the routine-basis experiment.

A part of this research was supported by a grant from the Ministry of Education, Sports, Culture, Science, and Technology (MEXT) of Japan through the NAREGI (National Research Grid Initiative) Project.

References

[1] I. Foster and C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *Supercomputing Applications and High Performance Computing*, vol. 11, no. 2, pp. 115–128, 1997.

[2] H. Takemiya, K. Shudo, Y. Tanaka, and S. Sekiguchi, "Constructing Grid Applications Using Standard Grid Middleware," *Grid Computing*, vol. 1, pp. 117–131, 2003.

[3] G. Allen and et al., "Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus," in *Proceedings of Supercomputing 2001*, 2001.

[4] J. Pruyne and M. Livny, "Managing Checkpoints for Parallel Programs," 1996.

[5] E. F. Graham and et al., "HARNESS and fault tolerant MPI," *Parallel Computing*, vol. 27, pp. 1479–1496, 2001.

[6] G. Bosilca and et al., "Mpich-v: Toward a scalable fault tolerant mpi for volatile nodes," in *Proceedings of Supercomputing*, 2002.

[7] Y. Tanaka, H. Takemiya, H. Nakada, and S. Sekiguchi, “Design, implementation and performance evaluation of GridRPC programming middleware for a large-scale computational grid,” *Fifth IEEE/ACS International Workshop on Grid Communting*, pp. 298–305, 2005.

[8] K. Seymour and et al., “Overview of GridRPC: A Remote Procedure Call API for Grid Computing,” in *Proceedings of 3rd International Workshop on Grid Computing* (M. Parashar, ed.), pp. 274–278, 2002.

[9] T. Ikegami and et al., “Accurate Molecular Simulation on the Grid – Replica Exchange Monte Carlo Simulation for c_{20} Molecule,” *Journal of Information Processing Society of Japan*, vol. 44, no. SIG11, pp. 14–22, 2003.

[10] K. Yabana and G. F. Bertsch, “Time-Dependent Local-Density Approximation in Real Time: Application to Conjugated Molecules,” *Quantum Chemistry*, vol. 75, pp. 55–66, 1999.

[11] “PRAGMA.” http://www.pragma-grid.net/.

[12] “ApGrid.” http://www.apgrid.org/.

[13] J. Goux, S. Kulkarni, J. Linderoth, and M. Yoder, “An Enabling Framework for Master-Worker Applications on the Computational Grid,” in *Proceedings of HPDC-9*, pp. 43–50, 2000.

[14] H. Nakada, Y. Tanaka, S. Matsuoka, and S. Sekiguchi, “The Design and Implementation of a Fault-Tolerant RPC System: Ninf-C,” in *Proceedings of HPC Asia*, pp. 9–18, 2004.

[15] H. Casanova and J. Dongarra, “Netsolve: A Network Server for Solving Computational Science Problems,” *Supercomputing Applications and High Performance Computing*, vol. 11, no. 3, pp. 212–223, 1997.

DMOVER: Parallel Data Migration for Mainstream Users

Nathan T.B. Stone, Bryon Gill, John Kochmar, Rob Light,
Paul Nowoczynski, J. Ray Scott, Jason Sommerfield, Chad Vizino
{stone, bgill, kochmar, light, pauln, scott, jasons, vizino}@psc.edu
Pittsburgh Supercomputing Center
4400 Fifth Avenue
Pittsburgh, PA 15213

Abstract

DMOVER is a combination of three highly portable scripts and a network optimization specific to the Pittsburgh Supercomputing Center (PSC). DMOVER provides users with a queue-oriented means for initiating large, parallel, inter-site data transfers that is expected to be familiar to all mainstream HPC users. Its flexibility enables users to effectively manage a parallel, load-balanced transfer of hundreds of files between dozens of file servers. It provided these highly-desirable features at a time when they were not present in other standardized grid services and which remain elusive on some platforms. Performance measurements consistently indicate that file transfers remain disk-rate-limited and that the DMOVER strategy of maintaining parallel streams is a highly efficient means for aggregating available resources.

1. Introduction

The Extensible Teragrid Facility[1] (ETF) is a massive collection of high-performance computing, networking and storage resources woven together by a software infrastructure designed to extend its functionality and ease of use to the current generation of scientific researchers. This substantial task has turned out an interconnected web of authentication & authorization, data transfer, job management and accounting software that can still confuse even informed users. Many developers within the ETF community have rededicated themselves to eliminating this confusion, redesigning some interfaces or providing additional tools to ease the learning curve for users in the hopes that this will deliver the promises of grid computing to mainstream users. "DMOVER"[2] is one such effort at the Pittsburgh Supercomputing Center (PSC) directed at facilitating large, inter-site, parallel data transfers.

1.1. Problem Summary

Data migration is a recurring challenge for most ETF users. Despite the presence of a high-performance network and file transfer middleware, moving thousands or even hundreds of files across the ETF network can still be a slow or tedious process for mainstream users – those with a lower tolerance for instability or infrastructural complexity. Such users are perceived to be dependent upon resources and utilities delivering exceptional performance (in this case, aggregate bandwidth) exactly as advertised every time. This is an obvious challenge to the resource administrators and middleware and system software developers.

Furthermore, a recent survey of Hierarchical Storage Management (HSM) utilization at PSC revealed that over 93% of data stored in our HSM are transferred in "sessions" involving 10 or more files. In such sessions the average number of files was 378, with an average file size of 93 MB. So if we are to adequately address the problem of transferring real datasets it must be in a context of facilitating transfers with large file count and moderately large files.

1.2. Solution Summary

DMOVER[2] is fundamentally an integration effort unifying three readily available resources on "LeMieux"[3], PSC's 3000+ processor ETF computing resource. First, DMOVER uses the batch scheduling system to allocate both dedicated file servers and Application GateWay (AGW) nodes (see Section 3.4). Second, it uses the standardized grid file transfer tools (*e.g.* `globus-url-copy`) to transfer files resident within LeMieux's global parallel scratch file systems. And third, it uses a transparent communication library called Qsockets to redirect TCP-based traffic from the default Ethernet network to the Quadrics QSNet-based computational interconnect and to the ETF network via the AGW nodes. DMOVER executes these transfers in parallel, managing a user-definable number of transfer streams until all designated files are transferred.

2. Problem Details

At PSC we were motivated by the expressed concerns of a particular user who had a clearly specified need: to transfer terabytes of data from PSC to the San Diego Supercomputing Center (SDSC) in a timely manner. (Transfer rates at the time, he complained, were typically of order 1-3 MB/s—more characteristic of an `scp` client than of any high-performance tool). If we failed to establish an effective channel then this user would simply not compute at our site. So we were motivated to write our own solution both to raise the performance and to "lower the bar", making it likely that this user would use our solution in this case and perhaps others.

We first surveyed the grid middleware tools at our disposal, listed following.

- The `globus-url-copy`[4,5] client even in recursive mode was clearly inadequate, due to the observed deficiency of single-stream bandwidth and the high file count involved. Other grid data transfer clients (*e.g.* `gsincftp`), while providing a slightly different interface, deliver no better performance.
- "Striped GridFTP"[6,7], a protocol for sending parts of a single file in parallel via multiple 3rd-party transfers, was not yet available at that time. At present it is still not available for Tru64 UNIX due to a reliance upon the Globus Toolkit V4.0[8], which does not exist yet for that OS. Nevertheless, the unsuitability of this protocol to the transfer of large file counts is confirmed by the command line switch required to activate it ("`-big`", pertaining to file size).
- The `uberftp`[9] client supports a parallel mode which is expected to be effective for 3rd-party transfers but of limited usefulness for direct transfers (client-based upload/download). The parallel streams in this case are implemented in such a way that each file is divided in <N> parts (where <N> is specified on the command line) but all transfers terminate at the client's host. This inherently limits the total aggregate bandwidth to the network connection of the client host, although it may benefit cases in which single-stream bandwidth has lower hard

limits.

- Reliable File Transfer[10,11] (RFT) has also been only recently deployed. It is generally well-suited to the parallel transfer of large numbers of files, and the administrator-configured number of parallel streams by default is of order 12 to 15. However, it relies on the presence of 3rd-party GridFTP servers for accessing all of the storage in question. As noted above, this alternative is unavailable within LeMieux, since it is running Tru64 UNIX.

If users are forced to write their own "helper scripts" or other tools in order to use our middleware they will simply not adopt it.

3. Solution Details

We resolved to build a light-weight parallel aggregation layer over `globus-url-copy`, to at least utilize its authentication scheme and tunable options. Our aggregator, now known as DMOVER, incorporated the following components at the PSC site:

1. Parallel file servers: LeMieux contains a pool of 64 file servers that export a global parallel file system. At present 16 of these are dedicated to a reserved "DMOVER" queue for allocation by the scheduling system.
2. QSNet[12]: LeMieux is interconnected by a high-performance Quadrics network called "QSNet", which routinely delivers DMA transfer rates of order 250 MB/s per NIC. The Application GateWay (AGW) nodes, HSM cache nodes, and Rachel (rachel.psc.edu) are also on the PSC QSNet.
3. Application GateWay (AGW) nodes[13]: PSC maintains a scalable number (currently 16) AGW nodes dedicated to relaying network traffic between the ETF network and PSC's internal QSNet. Each node has interfaces on the ETF network, the PSC LAN, and the PSC QSNet. Because there are a finite number of AGW nodes, we have configured these nodes are a scheduled resource, allocated by our batch scheduling system.
4. Qsockets[14]: Developers in PSC's networking group have created the "Qsockets" system, which unites a Qsockets server that serves as a data router and a client process via the QSNet. Qsockets provides a client library that intercepts normal TCP socket function invocations in legacy binaries running on LeMieux compute nodes and relays them over the QSNet to the Qsockets server resident on an AGW node. The server then passes the traffic on to the ETF network or whatever external network is most appropriate. In this way processes within LeMieux gain virtual interfaces to the ETF network.

Our solution requires a user to do three things:

1. Acquire the proper grid credentials (no different than for normal grid operations)
2. Specify source and destination by changing a few (4-6) lines in a batch script
3. Queue a job to a dedicated DMOVER queue (via the same queuing system with which they are already familiar)

The scheduling system itself allocates the requisite number of AGW nodes for parallel, multi-stream transfers. The DMOVER suite expands the source specification into a list of target files, builds the command lines, and launches multiple transfer streams in parallel, keeping a fixed number of streams running through the AGWs at all times until all transfers are completed. Below are details describing each of the components.

DMOVER consists of a trio of scripts: a batch script, a process manager and a transfer agent (command line wrapper). This solution required no modifications or additions to any other existing elements at any site.

3.1. The Batch Script

The first script in the trio is the DMOVER batch script (*e.g.* `dmover.pbs`). A user submits this script to the scheduler (OpenPBS) to initiate a transfer. The advantage of this approach is that batch scripts are highly familiar to all HPC users. This eliminates the usage barrier of interface style. Figure 1 below shows a sample job script that a user could queue to initiate DMOVER transfers to SDSC via 4 parallel streams.

```
#PBS -l rmsnodes=4:4
#PBS -l agw_nodes=4

# root of the file(s)/directory(s) to transfer
export SrcDirRoot=$SCRATCH/mydata/

# path to the target sources, relative to SrcDirRoot
export SrcRelPath="*.dat"

# destination host name
export DestHost=tg-c001.sdsc.teragrid.org,
tg-c002.sdsc.teragrid.org,tg-c003.sdsc.teragrid.org,
tg-c004.sdsc.teragrid.org

# root of the file(s)/directory(s) at the other side
export DestDirRoot=/gpfs/ux123456/mydata/

# run the process manager
/scratcha1/dmover/dmover_process_manager.pl
"$SrcDirRoot" "$SrcRelPath" "$DestHost" "$DestDirRoot"
"$RMS_NODES"
```

Figure 1: Sample DMOVER job script.

Number of Streams

Note first that the user specifies the number of streams for the transfer by setting both the number of nodes to use from the dedicated pool of file servers (`rmsnodes=4:4`) and the number of AGW nodes (`agw_nodes=4`). These directives are for the scheduling system, thus are prefixed with the "`#PBS -l`" resource specification. Per-stream performance is optimal when using one process per node (*e.g.* "`rmsnodes=4:4`") as compared to sending multiple streams from the same file server (*e.g.* "`rmsnodes=1:4`", representing a request for 1 node on which 4 processes will run). Furthermore, performance is best when the number of AGW nodes reserved matches the number of transfer streams. (See Section 5 below for a more thorough discussion of number of streams and performance.)

Source Path
The "`SrcDirRoot`" specifier identifies the absolute root path to the files to be transferred. This is a convenience that may or may not help the user in specifying "`SrcRelPath`". The "`SrcRelPath`" variable specifies the files and/or directories to be transferred relative to "`SrcDirRoot`" by name or by wildcard. In the case where there is a single directory tree to be transferred this is a straightforward process. But the value of `SrcRelPath` is expanded in the DMOVER process manager script by UNIX glob, which may in general match a large number of files and/or directory trees. The net result is that the process manager converts this information into a list of file targets to be transferred.

Destination Host
The destination is specified as a hostname string. In general that string can be a comma-separated list of valid hostnames. This was preferable over using some DNS-based host aliases because earlier versions of `globus-url-copy` would abort transfers in the case where the `gethostbyname()` invocation returned a server that was down as the first entry in the IP list. By allowing users to specify only servers that are up and available we enabled them to avoid known potential problems if they so choose.

This understated feature is of extreme relevance to bridging the gap between "early-adopters" and "mainstream users". There is often a disconnect between the tools used by these respective groups – one arcane version for early adopters and one "simple" version for mainstream users. The latter is often "simpler" because it is completely different, and many of the control interfaces are removed or obscured.

In this case we provide one single field for specifying a destination host. It can either be something "simple", like a single DNS-based host alias (*e.g.* `tg-gridftp.psc.teragrid.org`) or it could be a list of selected server names (*e.g.* `tg-c001.sdsc.teragrid.org, tg-c002.sdsc.teragrid.org,…`). But in either case the vehicle is the same, and the user can specify as little or as much detail as desired. This strategy effectively bridges the complexity gap by allowing either case, growing with the users as they grow their experience.

The next script level, the DMOVER process manager, will use a round-robin scheme to utilize all of the available hosts in turn for each successive transfer, thus achieving a basic level of load-balanced parallelism.

3.2. DMOVER Process Manager

As noted above, the process manager (`dmover_process_manager.pl`) parses and expands the values of the source and destination variables the user specifies in the batch script, building a list of specific transfers from these values. It then spawns an independent transfer for each case, launching a fixed number of transfers in parallel.

As earlier transfers finish the process manager selects another transfer from the list of remaining targets and starts it. Although the process manager itself runs on one of the OpenPBS "mom" nodes (currently configured as the LeMieux login nodes) the transfers are initiated remotely on the dedicated file servers allocated to the DMOVER queue.

This mode of launching many parallel transfers simultaneously is precisely what many users have been missing. This is contrasted with striped transfer, which is typically only advantageous for small numbers of very large files. To clarify, if all other transfer characteristics are equal, parallel streams will generally out-perform striped streams for cases where the number of files is much larger than the number of servers. And, as noted in the introduction, the average number of files transferred in a typical "session" is an order of magnitude greater than the number of file servers available at any site.

3.3. DMOVER Transfer Agent

Each of the parallel transfers is a single execution of the DMOVER transfer agent (`dmover_transfer.sh`). The transfer agent merely executes the Globus transfer client of choice (currently `globus-url-copy`). It uses the source and destination arguments supplied by the process manager and supplies these and other proper defaults to the file transfer client. But before it executes the Globus transfer it modifies the LD preload path (`_RLD_LIST`) to allow the otherwise "normal" TCP socket operations of the transfer client to be redirected over Qsockets, described in detail below. Using this technique we have benchmarked single-stream memory-only transfers across the ETF using `globus-url-copy` at roughly 1 Gb/s, or the theoretical maximum per-stream bandwidth for the hosts available.

The encapsulation of the final execution command line in the transfer agent script also provides a place for advanced users to optimize their transfer. For example users could select an alternative grid file transfer client (*e.g.* `gsincftp`) or customize their command line parameters to suit their needs by either modifying the provided script or substituting their own in the process manager.

3.4. AGWs and Qsockets on LeMieux

"LeMieux" is the largest compute resource on the ETF today. Its architecture leverages the high speed QSNet interconnect between nodes, but does not inherently provide for large bandwidths to remote hosts. Instead of connecting each compute node to the ETF network directly our design team introduced the concept of Application GateWay (AGW) nodes.

AGW nodes have network interfaces both on the ETF network (2 x 1 GigE NICs per node) and on the PSC QSNet network (1 QSNet NIC per node). The two-to-one matching between GigE and QSNet NICs led to a natural division of each AGW node into two "virtual" nodes, one serving each GigE interface. Each virtual node is equipped with a Qsockets "Qserver" that routes traffic from the PSC QSNet network (*e.g.* from LeMieux) to the ETF in a seamless manner. This configuration is so successful that an AGW node has no difficulty filling each of its GigE pipes from the QSNet, as described below. On the LeMieux side, Qsockets provides a client library that intercepts TCP-related system functions. In the case of setup/tear-down or ioctl functions, Qsockets merely passes these to the AGWs as requests over a QSNet-based RPC protocol. In the case of send operations the data is transferred over the QSNet to the AGW and from there routed to the ETF (and the converse for receive operations). In this manner processes running on LeMieux compute nodes can behave programmatically just as though they were directly connected to the ETF. This is all achieved without modifying any source or compiled code in the target application.

The efficiency and effectiveness of this strategy were proven during the “Bandwidth Challenge” during the SC’04 conference in November of 2004. In preparation for SC’04, PSC established a team whose purpose it was to design an end-to-end science demonstration suitable for the Bandwidth Challenge. The strategy of this team was to use a real scientific application that was instrumented to write its checkpoint data to a remote file server over a TCP connection. That much of the application had been created years prior as part of a simple checkpoint-recovery demonstration. For the Bandwidth Challenge, this application was redirected to transmit data not merely over Ethernet but over a combination of the QSNet and a custom 40 Gb/s link to the SC’04 show floor without changing any networking code in the application, but merely by setting the LD preload path as described above. Using this configuration that team sustained a write bandwidth of over 31.1 Gb/s end-to-end: from 32 LeMieux compute nodes, over 16 QSNet / dual GigE-connected AGW nodes, to 7 10GigE-connected file servers on the SC’04 show floor. This clearly demonstrates the high performance (high aggregate bandwidth) and low overhead (31.1 Gb/s over 32 GigE links on only 16 nodes) of the Qserver routing protocol on the AGW nodes.

4. Portability

The queue and process elements described above are completely portable. Every super-computing site has a scheduling system that can facilitate remote execution of multiple transmission streams on specialized nodes. Furthermore, the DMOVER scripts are written in Perl and Bourne shell, which is certainly accessible at all high-performance computing sites. The only non-portable aspect of this system is the AGW/Qsockets layer, which is a custom interconnect feature. Other sites have addressed their networking needs differently, for example by connecting compute nodes directly to the WAN. So the AGW/Qsockets layer does not require portability. If the DMOVER scripts were run at other sites mainstream users could simply comment out the AGW/Qsockets-related lines and run as at PSC.

Whether users would want to import DMOVER to a non-PSC site remains a valid question. Without DMOVER the only parallelism available to mainstream users is the number of GridFTP servers that have been configured by the site administrators. With DMOVER users have only to queue a DMOVER transfer job to a valid execution queue and they can utilize as many parallel streams as are available on that compute resource. This constitutes a perfectly reasonable utilization of HPC resources and a mainstream user’s compute allocation.

5. Performance

Below (Figures 2a and 2b) are performance measurements recorded for DMOVER transfers involving 32 files of size 2 GB (blue diamonds), 100 files of size 200 MB (red squares) and 300 files of size 100 MB (green triangles) from PSC (`/scratchb2`) to SDSC (`/gpfs`). These represent two reasonable test cases and a typical user storage scenario (as noted in Section 1.1), respectively. The transfers were performed varying the number of streams, and the bandwidths are reported as a function of number of streams. Figure 2a is a plot of the average per-stream bandwidth and Figure 2b is a plot of the bandwidth aggregated over all streams. At the time of testing the compute resource LeMieux was fully loaded. Thus the values represent a typical case for mainstream user activity. Indeed these results are consistent with lower values observed in other more optimal circumstances.

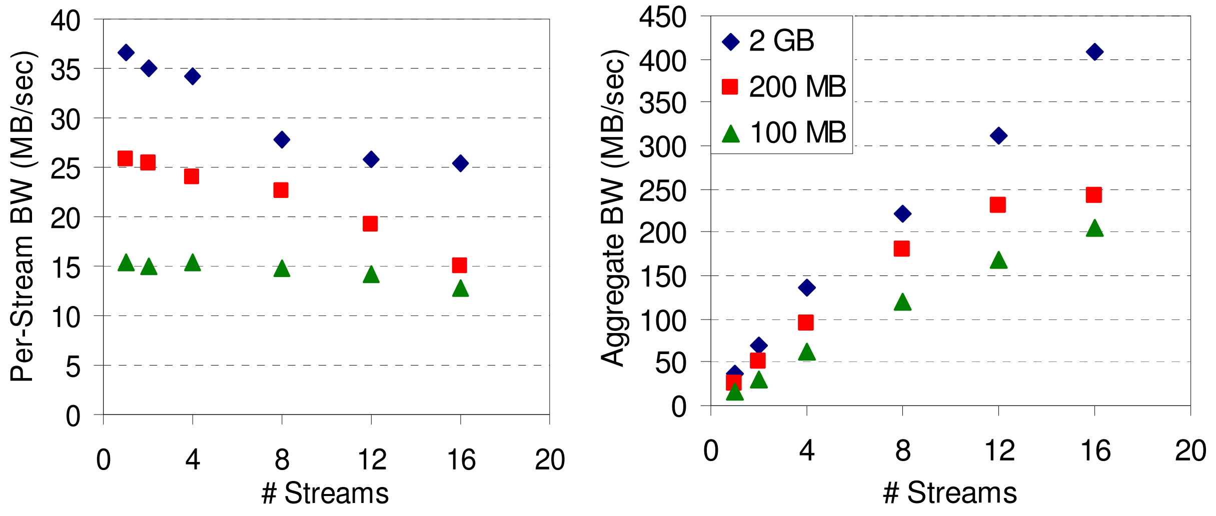


Figure 2: a) Average, per-stream bandwidth and b) aggregate bandwidth of DMOVER transfers for three file sizes, as labeled.

First we observe the trends in the data. In all cases we observe that the per-stream bandwidth for larger files is greater than that for smaller files. In fact, our 2GB test case achieves roughly double the bandwidth of the 100MB "average user" case. This exposes the extent to which the protocol overhead requires longer transfers to amortize the per-stream setup cost. We further observe that the per-stream bandwidth decreases with increasing number of streams. Thus the upward pressure to minimize aggregate transfer times is clearly at odds with the downward pressure of the scalability of the parallel transfers.

The aggregate bandwidth measurements are perhaps more encouraging. Even for the highest number of streams shown we do not appear to have reached a plateau or a point of diminishing returns for increasing stream count, despite the loss in efficiency. So we clearly demonstrate that users can easily achieve several hundred MB/sec aggregate with DMOVER by selecting the appropriate number of streams for their file size.

We continue by analyzing the host and network configuration underlying these results. At the sending side the number of hosts and associated GigE interfaces was matched to the number of streams, up to a maximum of 16. At the receiving side, the maximum was 12. Since the highest per-stream bandwidths observed up to 12 streams represent only 20-30% of the available network bandwidth to each host, the limitation is clearly not in the network. Furthermore, since (at least for cases with 12 or fewer streams) the transfer clients at each end are running on separate hosts we know that the processes cannot be adversely impacting one another. And yet the observed per-stream bandwidth clearly decreases with increasing stream count, even below 12. We conclude that the predominant bottleneck is in the parallel file systems at one or each end. To isolate this further (*e.g.* identify the limitations at each end) we would have to use the same transfer tool (`globus-url-copy`) to run device-only copies (*e.g.* from `/dev/zero` to `/dev/null`). This feature is not yet supported by `globus-url-copy` for `file://` type transfers. Device-only transfers between servers, however, have been benchmarked in excess of 110 MB/sec—wire speed for GigE-connected hosts.

Various groups have created ongoing ETF network and GridFTP diagnostics that continually monitor the performance of the network. One such effort is the PSC "SpeedPage".[15] This tool has indeed measured point-to-point single-stream bandwidths in excess of 90 MB/sec for some sites and file systems. This further supports the conclusion that investment in high-performance parallel file systems will make the greatest difference in inter-site GridFTP transfers.

6. User Experience

While the SC'04 Bandwidth Challenge well-established the efficiency and effectiveness of Qsockets for routing high throughput traffic from legacy applications, this is not sufficient to demonstrate its usefulness in the context of grid computing. The ultimate measure of the success of grid computing is the number of users and applications that use it to succeed in tasks that they could not have otherwise accomplished. The original incentive to create the DMOVER framework was the expressed need of an experienced user who wanted to migrate large amounts of data (of order Terabytes) in large numbers of files (of order thousands) from PSC to SDSC. Furthermore, he wanted to do this with regularity – not merely once or as a proof of concept. This motivated us to create an infrastructure to achieve substantial throughput from end-to-end between file servers at opposite ends of the country using Globus tools.

The resulting product is DMOVER, as described above. After using it himself our target user observed that this tool got him "past the Globus roadblock". He transferred Terabytes of data from PSC to SDSC at roughly 200 MB/s aggregate. So the parallelization model of DMOVER was precisely what enabled this user to achieve this level of aggregate performance.

7. Conclusions

DMOVER is a tool for initiating load-balanced parallel file transfers between sites on the ETF network. It achieves scalable aggregation of parallel streams thereby achieving a high effective bandwidth for large file-count transfers. Furthermore, this file-wise mode of parallelism is expected to be most applicable to typical user scenarios within the ETF community. DMOVER is an invaluable bridge between the raw grid services and the needs of mainstream users. Although the Globus ToolKit (GTK) incorporates similar functionality in RFT[10] even that utility is limited by administrative configurations (having sufficient GridFTP servers running wherever your data is staged) and code portability issues (GTK is not available on all platforms). If there are few or no servers running in such a location that they can effectively serve the file system of interest then mainstream users would be forced to resort to some other file transport system like DMOVER. Fortunately, with the portability and simplicity of DMOVER this should remain a viable option for mainstream users for the foreseeable future.

References

[1] The ETF (a.k.a. "TeraGrid") Project: http://www.teragrid.org/

[2] Usage documentation online at http://teragrid.psc.edu/lemieux/jobs.html#dmover

[3] Machine documentation online at http://www.psc.edu/machines/tcs/lemieux.html

[4] "Data Management and Transfer in High Performance Computational Grid Environments" B. Allcock, et al. *Parallel Computing Journa*l, Vol. 28 (5), May2002, pp. 749-771.

[5] Online documentation at http://www-unix.globus.org/grid_software/data/globus-url-copy.php

[6] "The Globus Striped GridFTP Framework and Server" Bill Allcock, John Bresnahan, Raj Kettimuthu, Mike Link, Catalin Dumitrescu, Ioan Raicu, Ian Foster. *Submitted to the 2005 High Performance Distributed Computing Conference (HPDC 14)*

[7] Online documentation at http://www-unix.globus.org/grid_software/data/stripedftp.php

[8] Private communication: Raj Kettimuthu of Argonne National Laboratory

[9] Software and online documentation at http://dims.ncsa.uiuc.edu/set/uberftp/

[10] "Reliable Data Transport: A Critical Service for the Grid" W.E. Allcock, I. Foster, R. Madduri. *Building Service Based Grids Workshop, Global Grid Forum 11*, June 2004

[11] Online documentation at http://www-unix.globus.org/grid_software/data/rft.php

[12] Product description at http://www.quadrics.com/Quadrics/QuadricsHome.nsf /DisplayPages/3A912204F260613680256DD9005122C7

[13] User documentation online at http://teragrid.psc.edu/lemieux/jobs.html#agw

[14] "Qsockets" M. Heffner, CMU Technical Report CMU-PSC-TR-2004-01, 2004 (online at http://www.psc.edu/publications/tech_reports/qsockets/qsockets.html)

[15] GridFTP performance monitoring at http://gridinfo.psc.edu/gridftp/speedpage.php

GeneGrid: Grid Service Based Virtual Bioinformatics Laboratory

P.V. Jithesh, Noel Kelly, Sachin Wasnik, Paul Donachy, Terence Harmer, Ron Perrott

Belfast e-Science Centre, Queen's University of Belfast
{ p.jithesh, n.kelly, s.wasnik p.donachy, t.harmer, r.perrott}@qub.ac.uk

Mark McCurley, Michael Townsley, Jim Johnston

Fusion Antibodies Ltd, Belfast
{mark.mccurley, michael.townsley, jim.johnston} @fusionantibodies.com

Shane McKee

Amtec Medical Ltd, Belfast
shanemckee @doctors.org.uk

Abstract

GeneGrid is a collaborative industrial R&D project initiated by the Belfast e-Science Centre, under the UK e-Science Programme, with commercial partners involved in the research and development of antibodies and drugs. GeneGrid provides a platform for scientists, especially biologists, to access their collective skills, experiences and results in a secure, reliable and scalable manner through the creation of a 'Virtual Bioinformatics Laboratory'. It enables the seamless integration of a myriad of heterogeneous applications and datasets that span multiple administrative domains and locations across the globe, and present these to the scientist through a simple user friendly interface. This paper presents how the grid services of GeneGrid are involved in the integration of bioinformatics applications as well as in the creation and execution of in silico experiments. A real use case scenario is also presented, involving the identification of novel members belonging to a protein family, for demonstrating the capabilities of GeneGrid. Experiences from the adoption of standards such as OGSA and the integration of third party programs, are also presented.

1. Introduction

Genome sequencing and post-genomic technologies such as microarrays, are creating an explosion in the number of biological datasets to be managed, integrated and analysed, pushing bioinformatics to the forefront of disciplines that need huge computing power and highly collaborative environments. The emergence of grid computing technologies has opened up an unprecedented opportunity for biologists to integrate data from multiple sources, in spatially distant locations, which can be seamlessly analysed leading to a greater chance of knowledge discovery.

GeneGrid is a UK e-Science industrial project with the involvement of companies interested in antibody and drug development. The aim of GeneGrid is to provide a platform for scientists to access their collective skills, experiences and results in a secure, reliable and scalable manner through the creation of a 'Virtual Bioinformatics Laboratory' [1]. GeneGrid accomplishes the seamless integration of a myriad of heterogeneous resources that span multiple administrative domains and locations and provides the scientist an integrated environment for the streamlined access of a number of bioinformatics and other accessory programs through a simple interface. It allows biologists to create, execute and manage workflows that represent bioinformatics experiments. Such workflows automate and hence accelerate the experiments, preventing errors that usually creep in because of manual interventions.

This paper presents the architecture of GeneGrid and its implementation based on the existing international standards. Experiences from developing Open Grid Services Architecture (OGSA) [2] based grid services using Globus Toolkit for the integration of bioinformatics applications and databases as well as the use of GeneGrid in the creation and execution of *in silico* experiments are discussed.

2. GeneGrid Architecture

GeneGrid consists of a number of cooperating Grid services developed based on the OGSA and using Globus Toolkit ver 3 (GT3). GeneGrid services may be categorised logically into different components, namely Workflow Management, Resource Monitoring & Service Discovery, Data Management, Application Management and the Portal, which are discussed below.

2.1. Application Integration

Access to the bioinformatics applications available on various resources is provided by the GeneGrid Application Manager (GAM) [3, 4]. GAM achieves this integration through two types of OGSA-based grid services: GeneGrid Application Manager Service Factory (GAMSF) and the GeneGrid Application Manager Service (GAMS).

GAMSF is a persistent service, which extends the standard interfaces or Port Types, like GridServiceFactory of the Open Grid Services Infrastructure (OGSI) [5] to integrate one or more bioinformatics applications to the grid, and exposes them to the rest of the GeneGrid. The primary function of GAMSF is to create transient instances of itself called GeneGrid Application Manager Services (GAMS) which facilitate clients to interface with the applications.

Any client wishing to execute a supported application will first connect to the GAMSF and create an instance - the GAMS. This newly created GAMS then exposes to the client the operations which allow the client to execute the supported application as an extension to the operations provided by the OGSA Grid Service interface. Each GAMS is created by a client with the intention of executing a given application, and after completion of this task the GAMS is destroyed. Currently GeneGrid integrates a number of bioinformatics applications including BLAST [6], TMHMM [7], SignalP [8], ClustalW [9] and HMMER [10]. In addition, GAM also integrates a number of custom programs developed to link the tasks in a workflow. Figure 1(a) gives an overview of the components that provide the GAM functionality.

2.2. Database Management

The GeneGrid Data Manager (GDM) is responsible for the integration and access of a number of disparate and heterogeneous biological datasets, as well as for providing a data warehousing facility within GeneGrid for experiment data such as results [11]. The data integrated by the GDM falls into two categories. 1). Biological data consisting of datasets available in the public domain, e.g. Swissprot [12], EMBL [13] etc. and proprietary biological data private to the companies. 2). GeneGrid data consisting of data either required by, or created by GeneGrid, such as workflow definitions or results information.

GDM has used OGSA-DAI (http://www.ogsadai.org) as the basis of its framework, enhancing and adapting it as required, such as for providing access to flat file databases. GDM consists of two types of services, replicating those found in OGSA-DAI. The GeneGrid Data Manager Service Factory (GDMSF) is a persistent service configured to support a single data set. The main role of the GDMSF is to create, upon request by a client, transient GeneGrid Data Manager Services (GDMS) which facilitate interaction between a client and the data set (Figure 1b).

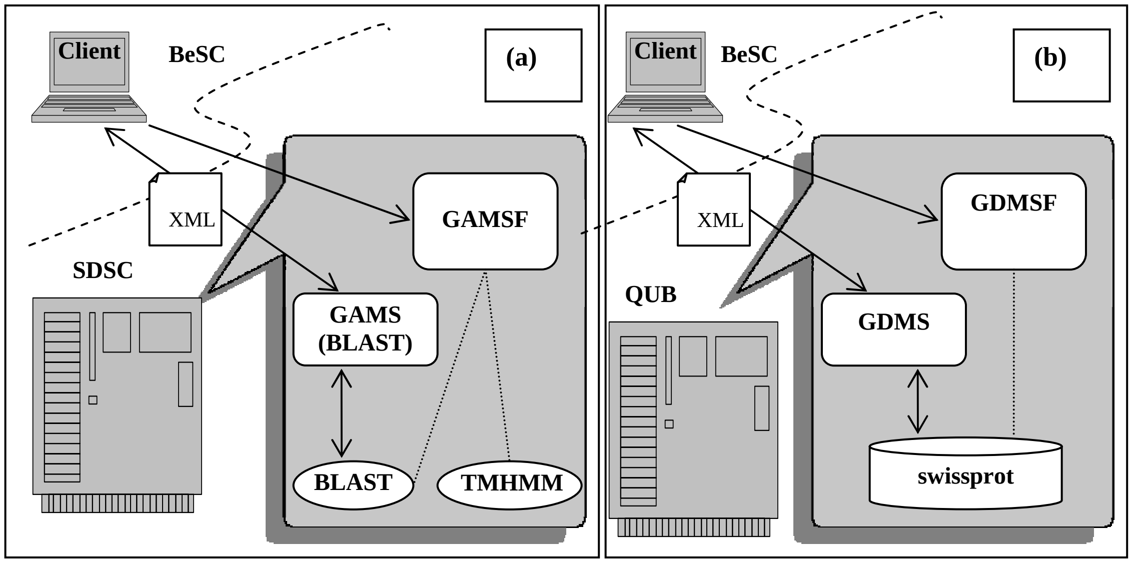


Figure 1. A client accessing (a) an application, e.g; BLAST on another resource through GAMS and (b) a database e.g: Swissprot through GDMS

2.3. Workflow Management

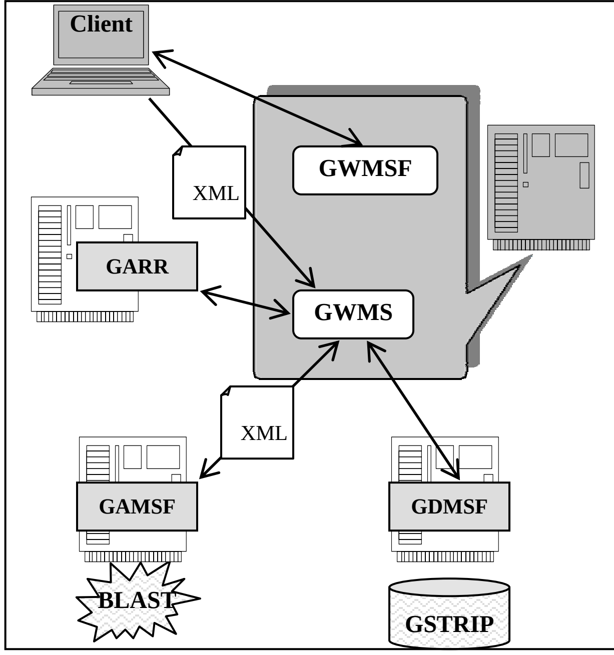
GeneGrid Workflow Manager (GWM) is the component of the system responsible for the processing of all submitted experiments, or workflows, within GeneGrid (Figure 2). As in the case of GAM, there are two types of services in the GWM. The first, the GeneGrid Workflow Manager Service Factory (GWMSF) is a persistent OGSA-based grid service. The main role of the GWMSF is to create GeneGrid Workflow Manager Services (GWMS), which will process and execute a submitted workflow across the resources available. Each GWMS is a transient grid service which is active for the lifetime of the workflow it is created to manage. The main roles of this service are to select the appropriate resources on which to run elements of the workflow, as well as to update the GeneGrid Status Tracking and Result & Input Parameters (GSTRIP) Database with all status changes. GWMS gets information on resources, databases, GDM services and GAM services through the GeneGrid Application & Resources Registry (GARR).

2.4. Resource Monitoring & Service Discovery

GARR is the central service in GeneGrid that mediates service discovery by publishing information about various services available in GeneGrid. A lightweight adaptor present on all the resources called GeneGrid Node Monitor (GNM) updates the GARR with the status of the resources, such as load average and available memory. In addition GNMs may also be configured to advertise details of the services deployed on the resources, such as service name, type, location and the database or application they integrate.

2.5. Portal

The GeneGrid Portal provides a secure central access point for all users to GeneGrid and is based upon the GridSphere product [14]. It also serves to conceal the complexity of interacting with many different Grid resource types and applications from the end users' perspective, providing a user friendly interface similar to those which our user community is already familiar with. This results in a drastically reduced learning curve for the scientists in order to exploit grid technology.



Submission and execution of a workflow containing a BLAST task. The client will first connect to the GWMSF to create a GWMS instance, before forwarding the workflow XML to the newly created GWMS. The GWMS identifies the BLAST task within the workflow, and queries the GARR for the location of all suitable GAMSF. The GWMS will then submit the task XML to the most appropriate GAM for execution. The GWMS will also find the location of the GDMSF serving the GSTRIP database from the GARR in order to submit updates as to the status of this BLAST task.

Figure 2. Workflow management by the GeneGrid Workflow Manager (GWM)

3. GeneGrid Component Integration

GeneGrid Environment (GE) is the collective name for the core distributed elements of the GeneGrid project, which allow the creation, processing and tracking of workflows. Contained within the GE is at least one GeneGrid Portal, at least one deployment of both the GARR and the GWMSF, an implementation of each of the GeneGrid Workflow Definition Database (GWDD) and the GSTRIP database, as well as at least one GDMSF configured to each of these databases. All instances of any factory services mentioned above may also be considered elements of the GE. By allowing users to access a GE, we create a Virtual Organisation (VO), and hence each GE may be considered as a single installation of GeneGrid.

Bioinformatics applications and datasets are exposed to the GeneGrid Environment by GAMSF and GDMSF respectively. These GAM and GDM services make up the GeneGrid Shared Resources. Each GAMSF and GDMSF advertises its existence and capabilities to a GE via GNM on their hosting nodes registering with the GARR. It is possible for GNM to register with many GARR services across multiple GE allowing the resources to be shared between multiple organisations. Therefore, organisations have complete control over what resources, if any, they wish to share with other GeneGrid organisations, forming dynamic virtual organisations.

4. GeneGrid Operation

Standard GeneGrid operation is workflow driven with scientists interacting with the system via the GeneGrid Portal to both generate and track workflows.

4.1 Workflow Creation

Having created GDMS for accessing both the GSTRIP and the GWDD, and having uploaded the Master Workflow Definition Document, the GeneGrid Portal is ready to create and submit new workflows. Users interact with the Portal to select required tasks, and fill out web based forms as presented by the Portal in order to generate a new Workflow XML document. The Portal will also upload information to the GSTRIP database as it is provided by the user. Once the user is happy with the workflow they have created, they may submit it for processing. The Portal will then connect to the GARR to retrieve the location of the GWMSF, and request it to create a GWMS instance. This newly created service is then sent the workflow XML for processing.

4.2 Workflow Execution

Having received the Workflow XML, the GWMS will proceed to break the workflow into its constituent tasks. The GWMS will connect to the GARR to find the locations of suitable GAM or GDM services capable of executing any tasks ready for execution. Tasks which rely upon the results of others are placed in a queue until such time as the information upon which they are dependant becomes available. The GWMS will also update the record for each task, and hence the workflow, within the GSTRIP database as appropriate.

4.3 Usability

GeneGrid development has seen a strong emphasis placed upon making the use of Grid technology as easy as possible for biological scientists. Currently, to perform an involved bioinformatics experiment with publicly available web sites would be a long tedious task with the user being asked to take a very “hands-on” approach at all times. This hands-on approach can be both time consuming and error prone. GeneGrid has automated this process considerably. Users may create a complex experiment from a single standard interface. As the linking of tasks together is automated, this considerably reduces the time required by the scientist to sit at a terminal! This point is emphasised further when we consider the recycling of experiment workflows by the end user to run the same experiment repeatedly with different input data each time. The absence of the need for manual intervention also reduces considerably the amount of errors that may “creep” into the process. Such automation has required the development of a number of custom “linker” applications for transforming the results of one application or database operation into something which can be understood by another. However, the GeneGrid Portal is designed to intuitively include such linker tasks into the workflow, allowing the scientists to concentrate on the applications and database with which they are familiar.

5. Use Cases

Scientists from the partner companies have tested the GeneGrid prototype by way of executing workflows which are biologically relevant. Such use cases have proved invaluable in providing feedback to the developers leading to bug fixes and further improvements. The use cases also set new requirements which led to the evolution of GeneGrid to a robust and versatile system. One such use case for the identification of novel protein family members is described briefly here.

Siglecs are a family of cell surface proteins belonging to the large Immunoglobulin superfamily, with a number of characteristic features shared among the members. They are involved in cell–cell interactions and signalling functions in the haemopoietic, immune and nervous systems [15]. The effort in this case study is to find new members of this promising family among the genomes using GeneGrid.

5.1 Use Case Workflow

In order to run such an experiment with existing technologies and resources would be a tedious time consuming task with quite a high risk of error. The scientist would take a known siglec sequence as the input for BLAST to find alignments, and when the results were available, extract all the accession numbers of interest to obtain the protein sequences. Each accession number would then have to be queried on line against the SwissProt database, with the resulting sequence being passed by the scientist into TMHMM. At this point, the experiment has forked from one experiment to multiple parallel experiments which must be tracked, and depending upon the BLAST configuration in the first step, the scientist could be tracking the progress of sequences numbering into the hundreds! The resulting TMHMM files must then be examined by the researcher, and if the file is of interest, the sequence used as input must be forwarded to SignalP, and again, the resulting files checked for success.

Using GeneGrid the scientist may create the experiment described above as a single workflow by supplying all the required input parameters. GeneGrid will automatically execute each task within the workflow once all the required data for that task becomes available. GeneGrid simplifies the experiment further by automatically including linker tasks on demand which are used to pre-process the results of one stage so that they are compatible with another e.g. BLAST results are processed to find all accession numbers before querying the SwissProt database.

This automation cuts the time required by the scientist to set up and track experiments considerably with GeneGrid managing the tracking of all experiment threads, and also eliminates the errors which creep in via manual intervention by the researcher. Finally, through the GeneGrid Portal, the scientist may track the progress of the experiment, examining the input and output files used at each point in the experiment.

Execution of the above workflow resulted in six uncharacterized and potentially new siglecs, which are currently being characterized using further procedures. Execution of the workflow, which would have taken about a day with conventional methods involving manual access to applications, took about 20 minutes in GeneGrid. This acceleration is largely due to the automation and parallelization of task execution, as well as the optimal use of available resources.

6. Discussion and conclusion

The use of grid technology and standards which are in the infancy has made GeneGrid development a challenging one. Furthermore, bioinformatics programs and databases usually have different proprietary formats and generally do not follow any standards. This has made the integration of multiple programs and data sources in GeneGrid quite difficult. A simple workflow which a biologist wishes to execute may not often suggest the underlying complexity in joining the tasks in the workflow.

The use of products from other projects also presented obstacles as those products are either in developmental stages or do not address GeneGrid’s requirements in its entirety. Sometimes such problems were alleviated by the extension of the third party products with custom solutions. For example, as GeneGrid required access to a number of flat file databases, OGSA-DAI product was extended to provide the required functionality.

The ability of GeneGrid to overcome these issues and provide automation of workflows shows distinctive advantages over conventional methods, a few of which are listed below.

- GeneGrid’s secure single access point provides users with a means to easily access many diverse applications and datasets without the need to visit many web sites.
- Automatic monitoring and selection of resources removes a major burden from the user, while ensuring an efficient allocation of resources.

- Considerably less time spent by the user creating and managing workflows.
- Errors which may creep in as a result of manual intervention are avoided.
- The ease of use of the GeneGrid front end means that scientists may exploit the promising potential of Grid technology while being insulated them from the inherent complexity of new underlying technology.

Thus, the development of a functional prototype of GeneGrid and its use in the problem of identifying new siglecs have clearly illustrated the viability of utilising grid services for integrating heterogeneous Bioinformatics programs with diverse requirements on different resources while following a workflow based approach.

7. References

[1] P. Donachy, T.J. Harmer, R.H. Perrott *et al*, “Grid Based Virtual Bioinformatics Laboratory”, *Proceedings of the UK e-Science All Hands Meeting (2003),* 111-116

[2] I. Foster, C. Kesselman, *et al.*, “The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration”, *Open Grid Service Infrastructure WG, Global Grid Forum (2002)*

[3] P.V. Jithesh, N. Kelly, D.R. Simpson, *et al* “Bioinformatics Application Integration and Management in GeneGrid: Experiments and Experiences”, *Proceedings of UK e-Science All Hands Meeting (2004),* 563-570

[4] P.V. Jithesh, N. Kelly, Paul Donachy *et al* “GeneGrid: Grid Based Solution for Bioinformatics Application Integration and Experiment Execution”, *IEEE Symposium on Computer Based Medical Systems, Dublin (2005).*

[5] S. Tuecke, K. Czajkowski, I. Foster *et al.,* Open Grid Services Infrastructure (OGSI) Version 1.0. *Global Grid Forum Draft Recommendation, (6/27/2003).*

[6] S.F. Altschul, *et al*, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs," *Nucleic Acids Res.,* vol. 25, pp. 3389-3402, Sep 1. 1997.

[7] A. Crogh *et al,* “Predicting transmembrane topology," *J.Mol.Biol.,* vol. 305, pp. 567-580, Jan. 2001.

[8] J.D. Bendtsen, H. Nielsen, G. von Heijne and S. Brunak, "Improved prediction of signal peptides: SignalP 3.0," *J.Mol.Biol.,* vol. 340, pp. 783-795, Jul 16. 2004.

[9] J.D. Thompson, D.G. Higgins and T.J. Gibson, "CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice," *Nucleic Acids Res.,* vol. 22, pp. 4673-4680, (1994).

[10] S.R. Eddy, “Profile hidden Markov Models,” *Bioinformatics,* 14, 755-763 (1998)

[11] N. Kelly, P.V. Jithesh, D.R. Simpson *et al*, “Bioinformatics Data and the Grid: The GeneGrid Data Manager”, *Proceedings of UK e-Science All Hands Meeting (2004),* 571-578

[12] R. Apweiler, *et al*, "UniProt: the Universal Protein knowledgebase," *Nucleic Acids Res.,* 32, D115-9, 2004.

[13] C. Kanz, P. Aldebert, N. Althorpe *et al*, "The EMBL Nucleotide Sequence Database," *Nucleic Acids Res.,* vol. 33 Database Issue, pp. D29-33, Jan 1. 2005.

[14] J. Novotny, M. Russell, O. Wehrens, “GridSphere: An Advanced Portal Framework”, *Proceedings of EuroMicro Conference (2004), 412-419*

[15] P.R. Crocker Siglecs: sialic acid binding immunoglobulin-like lectins in cell-cell interactions and signaling. *Curr. Opin. Struct. Biol.,* 12, 609-615 (2002).

From Proposal to Production: Lessons Learned Developing the Computational Chemistry Grid Cyberinfrastructure

Rion Dooley
Center for Computation and Technology
Louisiana State University
Baton Rouge, LA 70803
dooley@cct.lsu.edu

Kent Milfeld
Texas Advanced Computing Center
Commons Center 1.154D,
J.J. Pickle Research Campus
10100 Burnet Road (R8700), Building 137
Austin, Texas 78758-4497
milfeld@tacc.utexas.edu

Chona Guiang
Texas Advanced Computing Center
Commons Center 1.154D,
J.J. Pickle Research Campus
10100 Burnet Road (R8700), Building 137
Austin, Texas 78758-4497
chona@tacc.utexas.edu

Sudhakar Pamidighantam
National Center for Supercomputing Applications
605 E. Springfield Ave.
Champaign, IL 61820
spamidig@NCSA.UIUC.EDU

Gabrielle Allen
Center for Computation and Technology
Louisiana State University
Baton Rouge, LA 70803
gallen@cct.lsu.edu

Abstract

The Computational Chemistry Grid (CCG) is a 3-year, National Middleware Initiative (NMI) program to develop *cyberinfrastructure* for the chemistry community. CCG is led by the University of Kentucky, and involves collaborating sites at Louisiana State University, Ohio Supercomputing Center, Texas Advanced Computing Center, and the National Center for Supercomputing Applications. In this paper we discuss our experience developing the CCG cyberinfrastructure in the first year of the project. We pay special attention to sociological as well as technical issues we faced, and look forward to challenges we foresee in the remaining two years.

1 Introduction

The term *cyberinfrastructure*, coined by an NSF Blue Ribbon Panel, refers to software which enables scientists to exploit cutting edge technology resources, including compute and data servers, visualization devices, instruments and networks, for advancing our research in science and engineering.

The Computational Chemistry Grid (CCG) [1] is a three year NSF funded project to develop a cyberinfrastructure to serve scientists engaged in studying molecular structure and function.

Computational chemistry algorithms and software are now widely used across a broad range of disciplines, including nanotechnology, biotechnology, and material science. Around the world users of both commercial and academic chemistry software packages, such as Gaussian, GAMESS, MolPro and NWChem, are major users of both high and middle end compute resources. CCG will leverage existing established Grid middleware to provide an easy-to-use integrated computing environment for these and other chemistry applications, including allocations on computing resources.

CCG is led by the University of Kentucky (UKy), and involves collaborating sites at Louisiana State University (LSU), Ohio Supercomputing Center (OSC), Texas Advanced Computing Center (TACC), and the National Center for Supercomputing Applications (NCSA). In this paper we discuss initial experiences developing the CCG cyberinfrastructure through the first year of the project and look ahead to challenges in the remaining two years. This paper discusses the important issues faced, both technological and sociological, as well as the chosen solutions.

The remainder of the paper is as follows. In Section 2 we give an overview of the CCG and the GridChem client application. In Section 3 we discuss the challenges of implementing the technological infrastructure and the roadmaps developed at the start of the project. In Section 4, we discuss some of the sociological issues of providing a community infrastructure. Finally, we outline the current state of the CCG and show how the community is adopting this new infrastructure as a means of facilitating their research.

2 Overview

The design of CCG, as shown in Figure 1, is a 3-tier architecture comprised of a client side GUI application, a middleware service, and a resource layer. The client application, called GridChem, is an open source Java application that remotely launches and monitors computational chemistry calculations on CCG supercomputers at remote sites. GridChem is a "lightweight" application and is distributed as an "executable jar" file. In the current release, installation includes downloading the client and installing a server certificate that enables secure communication with the GridChem Middleware Server (GMS) for authorization requests and notification. In future releases, we will leverage the Java Web Start technology to wave the certificate requirement and automate the process on behalf of the user

As mentioned in Section 1, the goal of GridChem is to create a powerful and useful tool for the computational chemistry community that allows users to easily submit, monitor, and manage their jobs using a large set of existing computational chemistry applications on a broad set of resources. User interface design is a very important aspect of GridChem and a significant amount of effort has been spent addressing chemistry and application specific issues relating to the user interface. GridChem provides additional features such as application-specific molecular editors, output file parsing, and "hooks" to leverage customized visualization tools. More information on the GridChem client can be found at [1].

It was realized very early in the project that a robust, Service-oriented Architecture (SoA) would be needed to gain widespread acceptance by the user community. Problems of scalability, distributed resource management, and the fluctuating nature of the Quality of Service (QoS) provided by each resource are inherent in a static grid implementation. This is evident by the grid community's growing adoption of SoAs, such as WS-Is Basic Profile Compliant Web Services [2] , and the upcoming Web Services Resource Framework (WS-RF) [3]. In both of these SoAs, services may

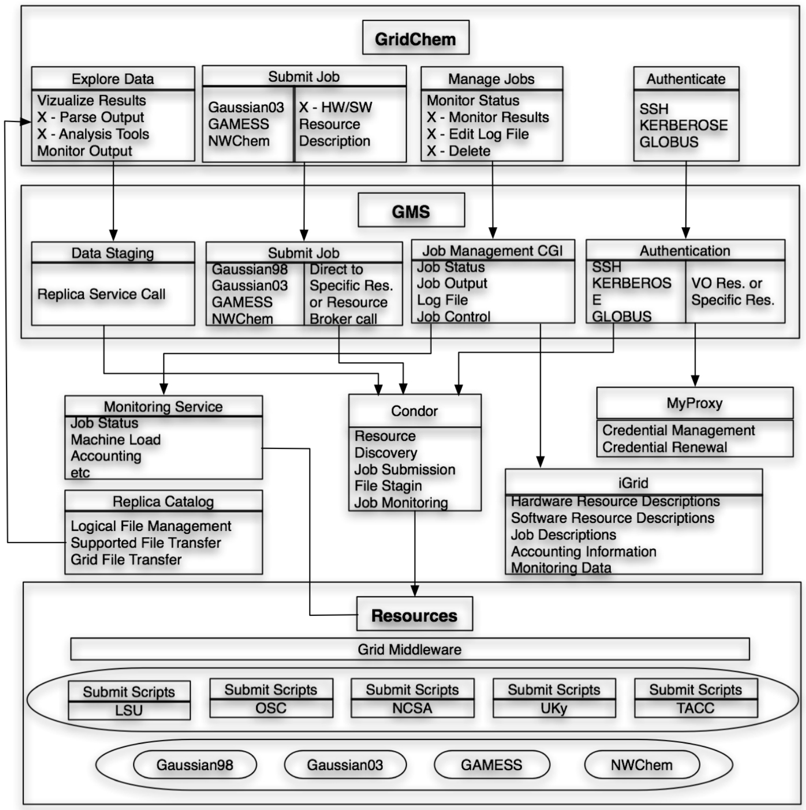


Figure 1: The planned architecture for the Computational Chemistry Grid. CCG is implementing a Service-oriented Architecture where the client utilizes the GMS for core functionality and the GMS in turn relies upon a series of grid and web services to provide functionality to the client. The current CCG architecture is very similar to the planned architecture. To move to this architecture, we will swap out existing CGI scripts for the GMS web services shown.

be composed hierarchically, allowing us to focus on developing the necessary meta-services needed for specific grid implementations, such as those in Section 3, rather than underlying low-level grid services. Using this approach, as the quality of the base services improves, so too will the quality of our meta-services.

Although the long-term goal of this project is to provide a robust, SoA, one of the primary deliverables in the first year of this project was to provide a production environment for users to submit, monitor, and retrieve output from jobs. To meet our short term goal of usability and to facilitate our long-term goal of implementing a robust grid architecture, we chose to first implement server-side functionality (ie. the middle layer of our architecture in Figure 1) in CGI scripts. Thus, the current middleware layer consists of basic grid middleware (Globus, NMI middleware distribution, etc.) and the client CGI scripts corresponding to individual client functionality. Using this approach, we can replace CGI scripts one-for-one with their corresponding web-service implementation as they become available. We choose web services to implement the future middleware layer instead of servlets or the existing CGI due to the complex nature of the final CCG architecture. As you will see in Section 3, the middleware must provide several complex features that would be difficult to achieve without heavy integration at the highest level. Using web services allows us to integrate the accounting, job submission, security, and monitoring components at a level not possible in any of the individual underlying services.

The lowest level of the CCG Architecture is the resource layer which appears at the bottom of Figure 1. The resource layer consists of the physical resources, local schedulers, resource-specific low level information providers, and the software needed to run the computational chemistry applications on each machine. Included in this layer is a set of lexical scripts, called by the CCG middleware, that collect resource information and place each job into the local batch scheduling queue. The job queueing scripts are application specific and tailored to each machine's unique characteristics. As the middleware services mature, the scripts will be phased out in favor of server-side resource brokering that will leverage robust software information provisioning.

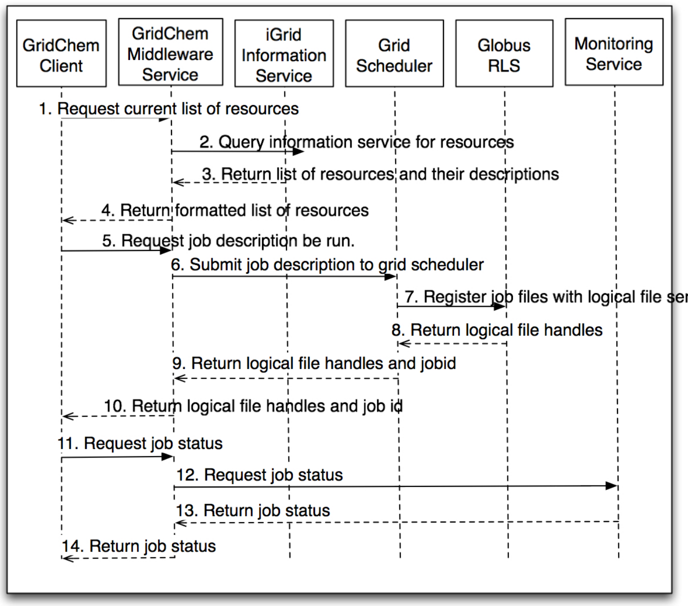

GridChem Client
GridChem Middleware Service
iGrid Information Service
Grid Scheduler
Globus RLS
Monitoring Service
1. Request current list of resources
2. Query information service for resources
3. Return list of resources and their descriptions
4. Return formatted list of resources
5. Request job description be run.
6. Submit job description to grid scheduler
7. Register job files with logical file service
8. Return logical file handles
9. Return logical file handles and jobid
10. Return logical file handles and job id
11. Request job status
12. Request job status
13. Return job status
14. Return job status

Figure 2: Sequence diagram of GridChem interaction with the GMS.

Further information on the interaction between the different CCG architectural layers is shown in Figure 2. Notice that each action in the client is reflected by a call to the corresponding GridChem Middleware Service. The GMS in turn, will leverage one or more underlying services to perform the requested action. In this manner, GMS takes advantage of continually maturing grid services without requiring significant development itself. More information on the CCG Architecture can be found at [4].

3 Technological Issues

Creating a production grid environment poses several significant technological problems related to security, software selection, account management, resource brokering, and accurate and up-to-date information provisioning. In this section we address each of these issues in turn.

3.1 Security

The CCG currently supports GSI, Kerberos, and Secure Shell security mechanisms. The decision to support these various mechanisms was a trade-off between wanting to develop a full grid architecture, and needing to gain wide community acceptance. Initially we decided to support all mechanisms and then gradually migrate users towards GSI security via our community account. The CCG community account is a multi-site allocation provided by each participating site to the community. Through the community account, we provide full GridChem functionality to users without requiring them to have an allocation on any individual machine.

Authentication and authorization both require significant planning. We first need to authenticate the user in order to map their (potentially numerous) userids to their unique GridChem account. This situation occurs when a user has multiple usernames on a machine and possesses more than one grid certificate that can authenticate. In addition to keeping track of the user's multiple identities, we must keep track of the authorization technique to use with each identity (ie. GSI, Kerberos, Secure Shell). Without such mechanisms, it is not possible to track account usage and determine the correct way to authorize the users on each machine.

Because we knew many users would not be familiar with grid technology, we could not require them to use their grid certificate for each operation they wished to perform. Additionally, we could not ask them to install a large, complicated suite of grid middleware simply to use the GridChem client. To avoid these issues, we allow users to authenticate to the GMS and perform all grid functionality, such as creating, pushing, and pulling a user's credentials from MyProxy [5], there, where the installation of middleware and details of executing grid functionality are handled on the user's behalf.

An example authentication session using the GSI mechanism is as follows. A user starts the GridChem client and opens the ”Authenticate” panel. There, they select ”Myproxy” authentication, enter their CCG username and password, then click on the ”Login” button. GridChem then encrypts the username and password and sends them as arguments to the GMS Authentication Service. The service checks to see if the user's information is correct, then pulls a community credential from our Myproxy server. Upon completion of this step, the user is authenticated and can use the full functionality of the client without thought for the underlying security mechanisms or having to authenticate again.

3.2 Accounting

Accounting is a fundamental concern for this project. Because we provide access to a community account, we must track not only overall account usage, but also individual user utilization. Further, we must solve each security scenario separately due to the unique characteristics of the job submission workflow related to each technique. For the grid scenario, we solved this problem by mandating individual user registration before granting use of the GridChem client application. By controlling all account allocations, when a user logs in, we can track their activity by mapping their chosen authentication technique to their CCG account using the DN from their grid certificate. We currently store this information in a server-side database. By doing this, we can manage individual accounts via a web interface or through GMS.

A problem which has proven to be much more difficult and remains an open question for this project is accounting for user activity when the user submits jobs outside of GridChem. One solution is to rely on regular parsing of the history records from the local schedulers and reliable queue monitoring. Often times, however this information is not readily available. Even assuming this information is available, we still then need to develop mechanisms to map local machine accounts to CCG users. When using GMS, we can locate user jobs by querying our database for using their unique user id. In more general cases, however, we do not have such a luxury. Reverse mapping is very difficult, if not impossible considering the user may use a special project account, a community account, or simply a personal account issued under a different site allocation to run jobs not affiliated with CCG.

To date, we have not found a definitive solution to this problem. Our primary focus at this time is verifying accurate accounting of GridChem jobs. Once we are able to guarantee that all jobs are audited correctly, we plan on incorporating information on the user's other jobs into the auditing process. However, as we mentioned above, it may not be possible to account for jobs not affiliated with GridChem. In practice, this issue may not prove to be a problem. Users may use GridChem for all their computational chemistry needs, or not hold GridChem liable for tracking their outside jobs. Without an active user community to examine the situation further, we cannot be certain. We expect that the general case will remain an open question until the project is well into its production phase.

3.3 Resource Brokering

As mentioned in Section 2, the problem we address with the CCG is to provide cyberinfrastructure for the computational chemistry community that enables them to submit and manage jobs using a select set of well-known applications. Narrowing our focus from the general case to this specific case of only supporting a few known applications simplifies the task of resource brokering. We know what applications our user community will employ, we know the finite list of dedicated machines on which the user will run these applications, and we are able to control the means in which this introduction will happen. Using this information greatly reduces the complexity of our task and allows us to again leverage the SoA paradigm to perform resource brokering at two distinct levels.

At the lowest level, we use grid schedulers to submit and manipulate the user's job on every resource. Existing tools such as Condor [6], the Grid Resource Management System (GRMS) [7], and GRAM [8][9] fit this description. In order to avoid dependence on any one scheduling service, we leverage the Grid Application Toolkit (GAT) API [10] which allows us to interchange grid schedulers without altering our code base. It also allows us to take advantage of the best features of each technology and provide an overall service that is more sophisticated than it's individual parts. A good example where we have the opportunity to leverage existing technology to solve low-level problems is proxy certificate management. Computational chemistry jobs can vary in length from a couple hours to many months. As job run time increases, so to does the possibility that a user's proxy will expire long before their job finishes. If the user's proxy expires, all grid-based output file transfer will fail due to an expired credential. Condor-G is a grid scheduler that performs credential management on behalf of the user. Thus in the case of long running jobs, as an alternative to generating a credential with an extremely long life, we are able to use Condor-G as the underlying grid scheduler to renew the user's proxy credential on their behalf.

At the highest level, we will provide sophisticated services to the user such as throughput scheduling, economic scheduling, job monitoring, and notification. Intelligent scheduling of jobs, using different criteria for optimality, is one of the second year goals of the project and is crucial in ensuring efficient use of grid resources. By definition, throughput scheduling seeks to maximize job throughput by minimizing job turnaround time. Aside from requiring dynamic information that reflects current resource utilization, throughput scheduling necessitates reasonable values of three parameters that determine total job execution time, namely queue wait, data transfer and application run times. To obtain estimates of these parameters (within some specified error bounds), our metascheduler will utilize a web services-based prediction toolkit that implements the instance-based learning (IBL) method pioneered by Smith [11].

The job scheduling, monitoring and notification services are made available through the rich resource descriptions pushed to our information service. Detailed software and hardware resource descriptions allow us to integrate accounting into the decision making process, which in turn, enables better decisions than could be made by a third-party broker.

With better information comes better accounting, with better accounting comes better brokering. As we mentioned in section 3.2, we are working to provide more mature information providers and an advanced accounting system. As of this writing, these systems are not in place, thus our current resource brokering capabilities are dependent on the strength of the underlying grid schedulers we employ: currently Condor-G and GRAM.

3.4 Information Provisioning

Accurate and dependable information provisioning is the largest single challenge of this project. Without reliable information from all aspects of the system, we cannot make the necessary and intelligent decisions on the user's behalf. In line with our SoA architecture, we adopted the iGrid information service [12] to supply resource information to the CCG. iGrid is a hierarchical information service shown to perform several times faster than the Globus MDS [13]. The basic installation of iGrid, however, did not provide resource descriptions that were descriptive enough to meet our needs. Fortunately, iGrid is extendable and by working closely with the iGrid development team we were able to create a resource description schema suitable for the CCG. This proved to be relatively straightforward, taking less than a week, however, creating providers to discover local resource information took significantly longer.

Over the last month we have created the Job And Machine Monitor Service (JAMMS). JAMMS is a Perl script that pulls information on queues, running jobs, processor and machine utilization, wait time, and history. JAMMS is run as a cron job at each site. By default, the script is set to run every 5 minutes as a local user (e.g. ccguser). The Perl script uses the Perl Database Interface (DBI) module for sending information to a MySQL data base (at www.gridchem.org). A PHP program is used to extract information from the database and present it to the user, through their browser.

JAMMS programs, called filters, execute batch utilities (for either PBS, LSF, or LoadLeveler), and extract (filter out) needed information. Two or three batch utilities might be invoked from within the Perl script to obtain the relevant information. For LSF, a single API program was developed to quickly extract all relevant information. We are currently in the process of integrating the iGrid API into JAMMS so it can serve as a provider to the iGrid information service.

4 Sociological Issues

Equally as difficult as the technological problems associated with building a production grid are the sociological issues. Accommodating individual site policies, putting software in place, administering the various resources in a consistent way, and finding ways to to avoid stepping on each others' toes are all obstacles to overcome before a grid can ever be deemed, ”production”-worhty. This section discusses how we have, in the past, and continue to address each of these issues.

4.1 Community Trust

Participating in a grid can prove difficult because it often involves compromise on security and policy issues. Often this means agreeing to place a certain degree of control over the way local systems are run in the hands of someone else. It is an issue of trust. We know our colleagues, but how well do we really know them? Do we trust them enough to blindly issue accounts? Do we trust them enough to commit time and manpower to install and maintain a predefined, and often changing, software stack? Do we trust them to play by the rules by which we play? In short, how can we make sure that we're good grid participants without leaving ourselves exposed?

In the CCG, we approach each of these issues with a healthy dose of open communication. We are fortunate enough to represent sites of sufficient size and manpower to carry out the project, yet comprise a group of people small enough to develop interpersonal relationships. Weekly Access Grid [14] meetings allow us to ”see” each other on a regular basis and connect visually with the faces and voices behind the words. We implemented a solid system of documentation and discussion using several media from the telephone to email and from instant messaging to group Wiki pages [15]. This redundant level of communication and familiarity reduces the amount of insecurity and hesitation that exists when ”strangers” are forced to work together.

Complementary to building trust relationships, we must build a level of certainty that people are capable of performing the task at hand and dependable enough to carry out their work to the last detail. Sometimes that means saying, ”I won't have time to do that.” or, ”I'm not sure how to do that, let me do some reading and get back to you.” Being honest and accurate in our estimates and asking for help when we necessary has allowed us to more effectively work as a team.

As is true in life, with trust and certainty come confidence in the integrity of our colleagues. Through positive experiences with each other, we are reinforced with the knowledge that we are working with a group of people ”just like us.” In short, by committing to be good grid participants ourselves, we create an community of people who want to be good participants themselves.

Of course, not everything always works out as planned. We all experience setbacks, miss deadlines, and inevitably have our own ”best ways” of doing things. The important thing in our experience has been a willingness to listen and think things through. We cannot afford to let our egos get in the way of the project at hand. Individually we all win some and lose some, but in the end the goal is to make the project successful. If that happens, then all of us are winners.

4.2 Proprietary Licensing

Not everyone supports the concept of ”sharing” resources. There are significant obstacles to overcome, both legal and historical, before industry accepts the notion that a site license means unlimited use at that site. Unique pricing, wildly varying negotiation experiences, and rejection were all situations CCG member sites experienced when trying to purchase the license required to run different Computation Chemistry applications. In the end we decided that not every site would provide a complete set of end user applications. We believe that, for our needs, it is sufficient that each applications is available at a subset of CCG sites. This solution still gives the users several options where they can run their jobs and allow sites who cannot negotiate acceptable contracts with software vendors to participate in the project.

5 Community Acceptance

Having already released the first version of our software at the GridChem Workshop in April, we are now benefiting from user feedback and input from the computational chemistry community. Through surveys given at the workshop and conversations with individual users, we found that many users were very enthusiastic about the notion of a community account for running their jobs. The SSH authentication interface was extremely successful and users stated that they would be willing to adopt using grid certificates through a similar interface - especially if it meant gaining access to additional compute time.

One recurring request from many users was incorporation of a workflow engine into the GridChem client. Several researchers are currently performing task farming and/or more complex jobs that GridChem could potentially support. Because this is a valuable feature driven by user requirements, we will incorporate this into GridChem, however not in the near term as this functionality requires adaptation at both the client and server levels and would be best incorporated at a later stage in the project.

The last request users made was for more user-friendly file management. Keeping track of user output files is not an easy problem. It boils down to the same issues we face with accounting. As a first step, we are integrating Trebuchet, developed as a part of the Open Grid Computing Environment (OGCE) project [16], to provide a GUI-based grid file browser for the users. This feature, however, will not be available until the next full release.

We would like to thank Ian Kelley and Jon MacLaren for thoughtful review, as well as Michael Sheetz and the UKy development team for their contribution to the GridChem GUI. This work was funded in part by the National Science Foundation, Award #0438312.

References

[1] GridChem Project WebSite
http://www.gridchem.org/

[2] Web Services Interoperability Organization Basic Profile Version 1.1, Final Material, August 2004. http://www.ws-i.org/Profiles/BasicProfile-1.1-2004-08-24.html

[3] The Web Services Resource Framework. http://www.globus.org/wsrf/

[4] K. Milfeld, C. Guiang, S. Pamidighantam, J. Giuliani. Cluster Computing through an Application-oriented Computational Chemistry Grid. Proceedings of the 2005 Linux Clusters: The HPC Revolution.

[5] Myproxy WebPage
http://myproxy.org

[6] D. Thain, T. Tannenbaum, and M. Livny. "*Condor and the Grid*", in F. Berman, A. J. G. Hey, G. Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*, John Wiley, 2003.

[7] GridLab Project Website
http://www.gridlab.org/

[8] Globus WebPage - GRAM Overview.
http://www-unix.globus.org/toolkit/docs/3.2/gram/key/index.html.

[9] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In D. Feitelson and L. Rudolph, editors, Job Scheduling Strategies for Parallel Processing (Proceedings of the Fourth International JSSPP Workshop; LNCS #1459), pages 6282. Springer-Verlag, 1998.

[10] G. Allen, K. Davis, T. Dramlitsch, T. Goodale, I. Kelley, G. Lanfermann, J. Novotny, T. Radke, K. Rasul, M. Russell, E. Seidel, O. Wehrens. "*The GridLab Grid Application Toolkit.*" HPDC 2002: 411.

[11] W. Smith. "Improving Resource Selection and Scheduling using Predictions," in Grid Resource Management. J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds). Kluwer Publishing, Fall 2003.

[12] iGrid Website.
http://www.gridlab.org/WorkPackages/wp-10.

[13] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, S. Tuecke. "*A Directory Service for Configuring High-Performance Distributed Computations*". Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing, pp. 365-375, 1997.

[14] Access Grid WebPage
http://www.accessgrid.org/agdp/

[15] Wiki WebPage
http://wiki.org/wiki.cgi?WhatIsWiki

[16] OGCD Website.
http://www.collab-ogce.org/nmi/index.jsp.

Solving "Hard" Satisfiability Problems Using GridSAT [*]

Wahid Chrabakh
University of California Santa Barbara
Department of Computer Science
Santa Barbara, CA
chrabakh@cs.ucsb.edu

Rich Wolski
University of California Santa Barbara
Department of Computer Science
Santa Barbara, CA
rich@cs.ucsb.edu

Abstract

We present the latest instantiation of GridSAT [5], a distributed and complete satisfiability solver that is explicitly designed to aggregate grid resources for application performance. GridSAT was previously shown to outperform the state-of-the-art sequential solvers. In this work, we explore the unprecedented solving power GridSAT enables through algorithmic and implementation innovations. We describe the implementation techniques that allow GridSAT to leverage a variety of high-end batch-scheduled resources, clusters, interactive workstations, and personal computing resources through autonomous scheduling, checkpoint scheduling, and work migration. These innovations have allowed GridSAT to solve a set of "hard" and previously unsolved industrial and community satisfiability problems. In addition to this new solution power, GridSAT also outperforms the otherwise highest performance general solvers on the annual SAT competition performance benchmarks.

Keywords: *Parallel, Distributed, Scheduling, Satisfiability, Computational Grid.*

1 Introduction

Grid computing [11] is an emergent field in computer science that focuses, in part, on the aggregation of geographically distributed and federated computational resources. These resource aggregations can be harnessed by grid applications to solve problems in science and engineering [21, 1] which require large computing power. Solving such challenging problems and enabling new scientific results is an integral part of the grid computing vision.

One such challenging problem is propositional satisfiability. This problem involves finding a set of binary assignments to variables that satisfies a set of constraints (i.e. makes a binary expression evaluate to "true"). The problem of solving satisfiability instances is important from both theoretical and practical perspectives and is, in general, NP-complete. In practice, many engineering disciplines require the

[*]This work was supported by grants from the National Science Foundation, numbered CAREER-0093166, EIA-9975020, ACI-0103759, and CCR-0331654.

solution to domain specific instances of satisfiability. Such disciplines include scheduling, model checking, security, Artificial Intelligence, software verification, and the the area of Electronic Design Automation (EDA) which includes circuit design [29], Field-Programmable Gate Arrays (FPGA) detailed routing [23], combinational equivalence checking [18] and automatic test and pattern generation [20].

Because satisfiability solvers [22, 12, 15, 3] have become more efficient, they are now widely used in many industrial and research settings. There has been an extensive research effort geared towards the development of gradually more efficient satisfiability solvers [22, 12, 15, 3]. These solvers use different techniques to navigate the entire search space of possible truth assignments for the variables of a given expression. The best (fastest and most comprehensive) of these solvers use *learning* optimizations that permit the search space to be "pruned" during execution. Learning [28] introduces new deduced propositions which improve the solver's efficiency by obviating subtrees in the space of possible variable assignments.

Because learning requires a large, centralized database of intermediate propositions to be searched and updated frequently, the best known solvers are sequential. These sequential solvers are characterized by heavy use of compute power (CPU) as well as the memory of the host machine as the database must be kept memory-resident (or the speed becomes unacceptably low).

Research in parallel solvers [5, 17, 30, 8] , shows that using a large pool of computational resources leads to better performance for most problems. The aggregate CPU power and memory of the hosts allows the solver to navigate the search space faster. Thus a computational grid populated by a a large pool of resources offers potential improvements in solver speed. With the exception of those results reported in [5], however, the fastest solutions to the largest number of problems is generated by sequential solvers [26, 25].

By carefully leveraging the resources in grid settings, our goal is to build a parallel and distributed satisfiability (SAT) solver that correctly solves previously infeasible industrial problem instances, the answers for which cannot be determined in any other way. Secondarily, we would like to be able to solve faster the problems that sequential solvers find feasible.

Our previous work with GridSAT [5, 4] demonstrates the latter. By dynamically acquiring and releasing resources under the control of an automatic scheduler, GridSAT improves the time-to-solution for various feasible SAT instances. Indeed, GridSAT outperforms the best-known solver on all problems that this leading solver can complete [26, 25]. We have also been able to use GridSAT to solve several previously unsolved problems using non-dedicated, wide-area grid resources. It is these new domain-science results, and the techniques we have employed to achieve them, that are the subject of this paper.

In particular, by combining different batch-controlled super-computers with interactive workstations and user desktop machines, we have applied GridSAT to *hard* SAT problems – ones that are not only unsolved but for which previous attempts at solution using other general techniques have failed. This pattern of combining different types of resources is new and different from that used by existing *parallel* SAT implementations [17, 30]. Moreover, we know of no *distributed* (i.e. network and/or grid enabled) SAT implementations, efficient or otherwise, at the time of this writing.

The resources in a computational grid may be of two different types: time-shared or batch controlled. In the case of time-shared resources the application will compete with other user applications running simultaneously on the host machine. However, since these resources are always available the application can continue to make progress. Other resources which are controlled by a batch scheduler, will participate intermittently in the application through some of their nodes. But these systems will provide significant compute power depending on the size of the application's request.

In order to enable a grid implementation of a SAT solver to use many resources simultaneously, we need to address two types of challenges. First the solver's algorithm needs to be modified so that it can run in parallel while ensuring that the parallel components cooperate to improve over-all efficiency. The second challenge is developing a framework capable of running the parallel solver in a very volatile computational environment.

Solving the above two problems was at the core of our methodology in designing the application components and their interactions. Implementing this methodology can be achieved by selecting suitable technologies. Examples of these technologies include those from parallel computing, which predate grid computing,such as MPI [9]. The more relevant technologies are those which were the outcome of grid-specific research projects such as Globus [10], Web Services [33] and related standards. We discuss in this paper the requirements imposed by the application's dynamic behavior and constraints on the technology so that a successful implementation is realized. We also describe the current design and implementation of the application.

We have developed GridSAT, a distributed satisfiability solver capable of running on a computational grid. GridSAT implements a parallel algorithm for solving satisfiability problems based on Chaff [22]. GridSAT distributes and shares the internal proposition database among processors in a way that takes advantage of dynamic resource performance predictions to achieve new levels of solver efficiency.

In this paper, we detail the current, most capable version of GridSAT. Our most recent improvements in the clause sharing and resource scheduling algorithms have made it possible to solve previously unsolved satisfiability problems from the field of FPGA routing as well as artificially generated benchmarks specifically design to foil automatic SAT solvers.

2 GridSAT: SAT Solver for the Grid

A satisfiability problem is expressed as a boolean formula over a set of variables. Most solvers operate on formulas expressed in Conjunctive Normal Form (CNF) in which an expression conjoins (logically "ANDs") a set of *clauses*, each of which may contain disjoined ("ORed") literals. A literal is either an instance of a variable (V) or its complement($\sim V$) and variables are boolean. A SAT problem instance is termed *satisfiable* if there exists a set of variable assignments that makes the formula evaluate to *true* where "true" corresponds to a boolean 1 algebraically. If such an assignment does not exist the the problem is declared *unsatisfiable*.

GridSAT is based on Chaff [22], a sequential SAT solver algorithm. Chaff, in turn, builds upon the Davis-Putnam-Loveland-Logemann (DPLL) [7] algorithm which solves a SAT instance by making a set of speculative variable assignments (termed "decisions") stored in a *decision stack*. When these decisions are propagated through the clauses they could lead to a cascade of *implications*. Implications are assignments of boolean values to different variables as deductive consequences of previous speculative decisions. These speculative decisions and the resulting implications may lead to logical conflicts – deduced contradictions in which a variable must take on both boolean values because of different clauses in the original problem. In Chaff, as well as other solvers, the performance of the algorithm is enhanced by using techniques for adding new deduced clauses after a conflict occurs. This technique is called *Learning* [27, 19, 28]. Using learning, the algorithm may generate a vast number of additional clauses during execution. These clauses consume memory, possibly overwhelming the capacity of the host, and also may slow the algorithm as they can add to the search complexity of the clause database.

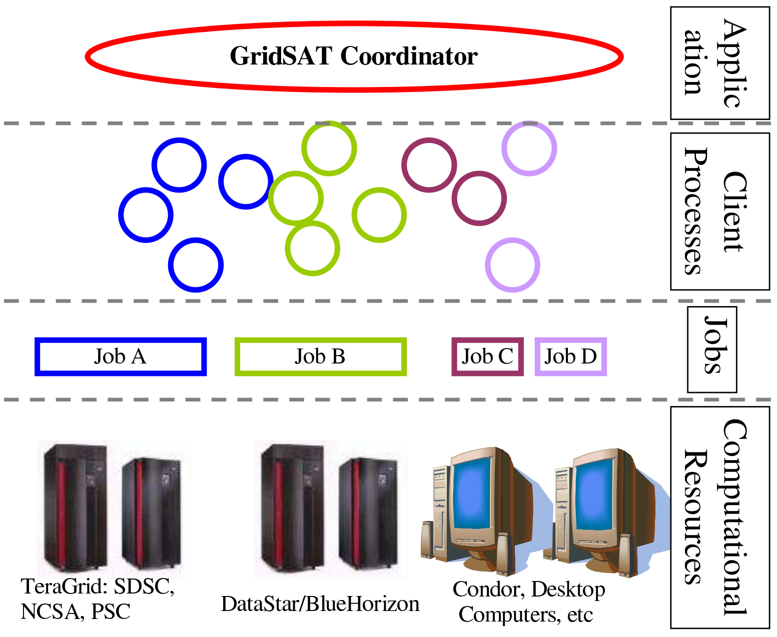


Figure 1. GridSAT resource views

GridSAT's distributed solver addresses three significant challenges to improving solver performance. First, GridSAT parallelizes the search algorithm that is navigating the space of possible truth assignments. Second, certain learned clauses from the various solvers are selected to be distributed and shared across resources. Finally, the GridSAT application components are dynamically scheduled at runtime to take advantage of those available resources which can enhance the solver's performance.

To apply a parallel search technique to SAT, we split the original problem into subproblems (having decision stacks with different truth assignments), each of which is independently investigated for satisfiability. Subproblems, themselves, may be split in the same way, forming a recursive tree, each node of which is assigned to a logically distinct processor. Clause sharing is facilitated by identifying and sharing only important clauses.

3 GridSAT Architecture and Resource Scheduling

GridSAT is implemented as a special form of the coordinator/client model where individual clients communicate directly and share clauses (i.e. communication is between peers rather than routed through the master). The GridSAT application uses two views of the computational resources as shown in figure 1. The first view employs jobs to classify processes which belong to the same resource. The second view is flat where all processes are part of a single pool. Both of these views are useful for managing resources under GridSAT

The coordinator (or master), shown in figure 2, reflects the resource views shown in figure 1. It consists of the resource manager, the client manager, the scheduler and the checkpoint server. We now describe the role of these components.

The resource manager is tasked with loading resource information from one or more grid information systems such as Globus MDS [6] and the NWS [37]. The scheduler, however, is responsible for coordinating the interactions between all the components. In addition, it handles interactions with external resources and monitors them to detect failures. For example, the scheduler queries the resource manager for resource types. If the resource is time-shared, then only one GridSAT process is launched. For batch systems, the scheduler instead submits one job request. Additional jobs could be manually submitted and GridSAT will use their resources when they become available. We term this form of scheduling *active queuing*; jobs waiting in queue logically execute on the interactive resources until the batch-controlled resources become available. At that time, the scheduler migrates work into the newly available resources. Thus, the application makes progress using the slower, shared resources while it waits in queue. It is the client manager that maintains a list of all GridSAT processes (active and queued) and monitors their progress.

The GridSAT scheduler is the focal point and is responsible for coordinating the rest of the components and launching new processes, also termed clients. The scheduler uses a progressive scheme for starting additional clients on remote resources and adding them to the active resources' pool. Resources which are no longer performing a task on behalf of GridSAT are released immediately when possible. The reason for this approach is the variability and unpredictability of resource usage for a particular SAT problem. Some problems are solved easily using a single host after a short time period. Other problems, however, might be harder and require a large number of hosts and a longer time period. By starting with a small resource pool and expanding the set of used resources, GridSAT achieves three goals. First, a small number of resources will be used to solve the easy problems which results in a smaller communication overhead and therefore shorter time to solve the problem. Second, GridSAT can adapt resource usage to how difficult the problem is perceived. If at a particular stage the problem is perceived difficult, the size of the resource pool used will grow. At another stage, the same problem might be perceived to be easy, a smaller resource set will be used, and excess resources will be released. Lastly, by remaining as small as possible at any given point in the execution, GridSAT promotes allocation stability and sharing. The scheduler does not waste resources needlessly thus the maximum number of GridSAT instances can co-exist since each is attempting to use as few resources as possible for its own problem instance.

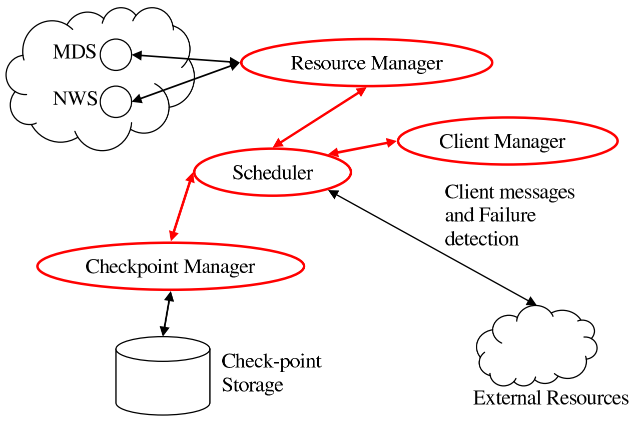


Figure 2. GridSAT components and their internal and external interactions. The external components and systems which GridSAT uses, such as the Globus MDS and the NWS, are shown in clouds.

The GridSAT scheduler uses the first available client immediately to start solving the problem. Each client records the time it took to receive the problem data. Clients also monitor their memory usage. The decision for splitting a problem is made locally by the client and not by a centralized scheduler. A client notifies the master that it wants to split its assigned subproblem with another client when its memory usage exceeds a specified limit (currently 80% of available memory) or after running for a specific period of time. This time period is determined as twice the duration of the communication period the client used to obtain the problem data. Using this method, the scheduler allows for computation time to offset the communication overhead by using the previous communication period as a prediction of future overhead. The clients, therefore, do not spend most of their time splitting instead of doing useful computation. The splitting process is performed by the cooperation of the master, the splitting client and an *idle* client. The *idle* client is a process which is not currently assigned a subproblem to investigate.

The GridSAT solver terminates when all subproblems are solved or one of the clients finds a satisfying assignment. In the latter case the client which finds the satisfying assignment sends its stack to the master. Finally, the master saves the final solution, terminates all running clients and cancels any pending resource requests. Most solvers in the literature are evaluated based on the time the first satisfiable instance is found. However there are cases [16] where knowing all satisfiable instances is helpful. GridSAT can also enumerate all the instances where a problem is satisfiable.

3.1 Active Queuing: Efficient Use of Batch Jobs

In GridSAT, initial batch job requests are large with a high number of nodes and long duration. This leads to a long waiting period in the scheduler's batch queue. Thus, if a job is not solved after this long waiting period, then it most probably is a hard problem. Therefore batch jobs are only used when the problem is hard. When a batch job starts execution, GridSAT migrates work (as a checkpoint file) to achieve more efficient use of batch nodes. Remote GridSAT nodes, which are numerous, will migrate immediately to occupy batch nodes. After, migration takes place and since networks are fast within super-computing nodes, splitting happens at higher rates especially after the above mentioned reductions in communication overhead. Moreover the GridSAT scheduler senses the additional bandwidth between clients executing on a supercomputer or cluster. It then increases the size and number of clauses shared by subproblems inside the tightly coupled resource as a further improvement. Note that the number of active nodes (i.e. those with subproblems) will increase exponentially. This happens because the number of new subproblems is increased in proportion to the number of existing active solvers. Problem migration leads to a more efficient use of batch jobs.

4 Grid Implementation

4.1 Application Characteristics

The GridSAT application is different from most high-performance computing applications. In general, these applications are composed of alternating steps involving computation and communication. The computation and communication intervals do not overlap. Also the communication steps are used as synchronization barriers which enable the various components of the application to exchange information. Moreover, these applications use a predetermined set of compute resources throughout their execution.

Our application differs in much of the above aspects. The GridSAT application has variable resource requirements depending on the problem instance. The number of resources and duration of use of those resources cannot be predicted in general for satisfiability instances. In fact, the set of active resources which are assigned parts of the search space during runtime is dynamic. Resources are added each time the problem is split. Also resources are released immediately after a subproblem is solved. There can be many simultaneous acquiring of new resources, through problem splitting, and release of other unneeded resources at any given instance. Moreover, the application components shared intermediate results as soon as they are produced. These results are asynchronously used by all the receiving clients.

Therefore, all the GridSAT segments are event driven and events are produced and consumed asynchronously. The solver components, for instance, can simultaneously perform communication and computation. All application modules are designed and implemented to allow for efficient management and responsiveness to these events.

Dynamic resource usage are needed, in general, to efficiently solve any satisfiability problem [5]. Solving "hard" satisfiability problems represents further challenges. For "hard" problems, a small number of resources would be exhausted in a relatively short time. The CPU and memory resources would be saturated and additional resources are required in order to make progress in solving the problem under investigation. Therefore, we wanted to use all computational resources at our disposal, in order to render the solution of the hardest problems more plausible. The set of available resources varied from desktop machines, to small-size clusters, to supercomputers. This collection of resources was heterogeneous in terms of hardware, Operating Systems and resource management software. This heterogeneity represents a further challenge to the deployment of the application.

These application characteristics described above represent a true Computational Grid application. Moreover, these characteristics are not unique to GridSAT. Other branch-and-bound or coordinator-worker applications can benefit from a similar use of computational resources.

A major challenge before implementing the various application components was to develop an implementation strategy. The final implementation aims at using all the available grid resources efficiently while dynamically adjusting to the application behavior and resource needs.

4.2 Implementation Strategy

Given these resource usage patterns, which are typical for a true Grid application, we had to choose an implementation strategy which would satisfy these requirements. There are several technology choices to select for the implementation of the application. Such options include, among others, MPI [9], Globus [10], vanilla Web Services [33] and later improvements such as WSRF [24].

According to our experience with GridSAT we have learned that a successful implementation technology should allow for three pivotal capabilities: dynamic resource pool, error detection and universal deployment.

The first capability is to allow the use of dynamic a resource pool. This feature, for example, was not available in MPI-I which did not allow for dynamic Communicators. MPI-2 has introduced extensions to allow for dynamic creation and destruction of communicators. Globus and Web services also allow for a dynamic set of resources.

The second capability is error detection and reporting. Since GridSAT runs for extended periods of time using a set of geographically distributed resources, then network and resource failures are more frequent. Therefore in order to implement this application we need a technology which allows for the detection of these errors. From the perspective of the application, the distinction between resource and network failures is not important. It suffices for the application to obtain a feedback if a certain operation is not successful after a certain time period.

Error detection and recovery is very important because in our experience all resources experience a failure at some point. Even those resources which are professionally maintained can become unresponsive from the application's perspective. Those resources that do not experience hardware and software failures usually have scheduled routine preventive maintenance periods or a combination of software and hardware upgrades. From the point of view of the application these are "scheduled" or "anticipated" failures. Without rigorous error handling the application would not be able to run for extended periods as shown later in the results section.

Different technologies provide some form of error handling. MPI-I allows for error handling in a limited scope which is expanded further in MPI-2. Globus GRAM allows for error handling and call-back functions for job management. In Web Services, WS-Notification [14], WS-BaseFaults [34] and related standards could be used to provide this functionality. The desirable error handling for our application is to provide a time period for some actions after which some form of error handling should be performed. Sometimes if an action fails, then all is needed is to retry it. In other cases, it is assumed that the resource (or the connecting network) has failed. This form of error handling is not available for the grid technologies mentioned above and can be implemented at the application level.

The last desirable capability for a suitable grid technology is universal deployment. This is not entirely a characteristic of the technology but of the computational environment as well. A widely deployed technology is advantageous because it reduces the development overhead since one version can be deployed on all available resources.In our experience there was no grid technology that was universally adopted and deployed which would enable us to combine all computational resources at our disposal.

Furthermore, in order to deploy our application over a large set of resources,we had to interface with many types of resource managers. For example, resources could be managed by one of many Batch schedulers, Condor [31] or simply shared. Our goal was to use all these resources simultaneously regardless of what systems they originate from. This is accomplished by determining a general job description which can be instantiated differently using specific launchers for each resource manager. For instance, shared resources can be accessed directly using SSH. Batch systems, however, are accessed by submitting a batch script with syntax tailored to the scheduler used. Whenever, Globus is deployed we use it to launch and monitor job submissions.

4.3 GridSAT Implementations

We believe that many of these technologies could be used to develop GridSAT. In fact, we have developed a previous versions of GridSAT called GrADSAT [4] (note the “A” in the spelling) using *GrADSoft*. GrADSoft is a set of programming abstractions where the baseline grid infrastructure is provided by Globus and the NWS. GrADSoft is part of the **Gr**id **A**pplication **D**evelopment **S**oftware (GrADS) project [2, 13] which is a comprehensive research effort studying grid programming tools and application development. To facilitate experimental application research and testing, the project maintains a nationally distributed grid of resources for use as a production testbed. Since the GrADS tools were universally deployed on this testbed we were able to deploy our application with little overhead on the entire testbed.

The current version of GriDSAT uses EveryWare [36] a very portable communication library. EveryWare has been designed explicitly to manage the heterogeneity and dynamism inherent in grid resource environments. EveryWare can be easily deployed as library on all the resources. In addition, all communication calls use a timeout argument, as desired, for error detection.

The resource management system interfaces with resources which use batch systems as well as desktop machines which are accessed through SSH. All resource related operations have been implemented to allow for a specific timeout. If the resource is not responsive after the timeout period expires, then the resource is considered unreachable.

In the future, we will explore other technologies as they become more widely used. Our goal would be to make GridSAT implementation independent where we can use an API for interfacing the application with the underlying communication infrastructure. As a result different grid technologies can be substituted without affecting the application.

5 Experimental Apparatus and Results

Since GridSAT is a true grid application, (robust, portable, heterogeneous, pervasive, etc. [11]) we ran a set of experiments to show that GridSAT can run for extended periods of time robustly using a wide variety of resources and also solve previously unsolved hard satisfiability instances. In these experiments we simultaneously use computational resources that belong to collections of individual machines, small size research clusters and super-computing scale clusters. The computational resources we use are composed from four main sources:(1) 40 machines from the VGrADS [35] testbed located at UTK, UCSD and UCSB, (2) Blue Horizon at SDSC, (3) TeraGrid site at SDSC, (4) TeraGrid site at NCSA and (5) DataStar at SDSC.

The TeraGrid [32] project is a multi-site national scale project which is aimed at building the worlds largest distributed infrastructure for open scientific research.

During our experiments, none of the resources we used were dedicated to our use. As such, other applications shared the computational resources with our application. It is, in fact, difficult to determine the degree of sharing that might have occurred across all of the available machines after the fact. In batch controlled system such as Blue Horizon, Data Star and the TeraGrid, the queue wait time incurred is highly variable because of jobs submitted by other users.

Thus, if it were possible to dedicate all of the VGrADS resources to GridSAT, we believe that the results would be better. As they are, they represent what is currently possible using non-dedicated Grids in a real-world compute setting.

These experiments also use a more diverse set of resources for longer periods of time (up to a month in duration) and multiple job requests. We chose a set of challenge problems from both SAT2002 [25] and SAT2003 [26] benchmarks. These benchmarks are used to judge and compare the performance of automatic SAT solvers at the annual SAT conference. All the problems in the benchmarks are shuffled to insure that submitted benchmarks are not biased in favor or against any solver. These benchmarks are used to rate all competing solvers. They include industrial and hand-made or randomly generated problem instances that can be roughly divided into two categories: *solvable* and *challenging*. The solvable category contains problem instances that some SAT solvers have solved correctly. They are used for comparing the speed of competing solvers. Alternatively, the challenging problem suite contains problem instances that have yet to be solved by an automatic method or which have only been solved by one or two automatic methods, but are nonetheless interesting to the SAT community. Some of these problems have known solutions that are known through analytical methods (i.e. the problem has a known solution by construction), but several of these problems are open questions in the field of satisfiability research.

In these experiments, we only chose problems from the challenging set. These problems were deemed hard by all participating solvers in both the 2002 and 2003 SAT competitions. We investigate seven previously unsolved problems where three instances are from the SAT 2003 benchmark category, and four are instances from the SAT 2002 benchmark category, all of which we have not been able to solve using previous versions of GridSAT.

This group of problems represent a variety of fields where problems are reduced to instances of sat-

File name	Description	SAT/UNSAT/*	Time	GridSAT Result
3bitadd-31.cnf	theoretical	UNSAT	8 days	-
k2fix-gr-rcs-w8.cnf	FPGA Routing	*	83261 sec (23 hours)	UNSAT
k2fix-gr-rcs-w9.cnf	FPGA Routing	*	14 days and 8 hours	UNSAT
cnt10.cnf	Theoretical	SAT	13134 sec (4hours)	SAT
comb1.cnf	Model Checking	*	11 days	-
f2clk50.cnf	Model Checking	*	9 days	-
hanoi6.cnf	Theoretical	SAT	23 days	-

(*): problem solution initially unknown

Table 1. GridSAT results using VGrADS testbed, Blue Horizon, Data Star and TeraGrid. All these problems were not previously solved by any other solver.

isfiability and solvers are used to determine the solutions. The problems contain a pair of problems in FPGA routing and model checking. These two disciplines benefit heavily from efficient SAT solvers. The remaining problems are of theoretical nature. In addition, we set the absolute minimum size of shared clauses to two and absolute maximum to 15. This range allows for sharing clauses which would help prune the search space without significant communication overhead.

Unlike previous experiments there was no timeout value set for the maximum execution time. Every problem was run using different job description for the batch systems. Jobs on the different batch queues were manually re-launched at random intervals. Job re-submission could have been automated but we wanted more control over rationing our limited compute budgets to specific experiments based on their perceived progress. Experiments where GridSAT was making progress were allotted bigger jobs with longer durations and more nodes. The progress of the solver was judged by inspecting how often the checkpoints were updated. We can also inspect the internal state of a particular solver using some of the tools we developed. The VGrADS nodes were used during the entire duration of each experiment unless the hosts experienced failures.

5.1 Results

The experimental results are summarized in Table 1. The first column contains the problem file name. The second column indicates the field from which this problem instance in obtained. The third column contains the solution to the instance: satisfiable (SAT), unsatisfiable (UNSAT), or unknown. We have marked those problem instances which were previously open satisfiability problems with an asterisk (*). If a problem was originally unknown and was later solved by a solver, then we still keep it marked with an asterisk for completeness. The fourth column represents the total wall-clock time that the problem was tried. Finally, the fifth and last column represents the solution obtained by GridSAT which is represented by SAT, UNSAT or (-) if we terminated the experiment before GridSAT found an answer. Note that while we terminated these problem instances manually so that we could complete this paper, each can be continued from its last checkpoint (which we have archived).

Table 1 shows that GridSAT was able to solve three problems all of which were not previously solved. Two of the problems were found unsatisfiable and they are both from the field of FPGA routing. The first problem *k2fix-gr-rcs-w8.cnf* was solved using the VGrADS testbed only. Batch jobs which were submitted for this experiment were canceled when the problem was solved. On the other hand the second problem *k2fix-gr-rcs-w9.cnf* took much longer to solve, it took more than two weeks. Table 2 gives a more detailed description of the resource used during this experiment. For each job a number of GridSAT solver components were launched as indicated in the last column of table 2. In table 3 a break down of the CPU-hours used on each resource are tabulated. Note that the VGrADS testbed machines were able to deliver a sizable amount of compute power because they were available in a shared mode for the duration of the experiment.

The last problem *cnt10.cnf* was also solved using the VGrADS testbed only under similar circumstances to *k2fix-gr-rcs-w8.cnf*. We previously tried solving this problem in [5] using the same testbed for four days in addition to Blue Horizon for 12 hours but were not successful. We believe the improvements made to the solver and especially the new clause sharing method have helped achieve this result.

Compute resource	Job count	Job dur.(hr)	Node count	procs /node
BlueHorizon	2	10	100	3
Blue Horizon	1	12	100	3
DataStar	2	10	8	11
TG@SDSC	1	10	40	2
TGd@SDSC	1	12	40	2
TG@SDSC	3	10	4	2
TG@SDSC	4	5	4	2
TG@NCSA	3	10	4	2
TG@NCSA	4	5	4	2

in addition to 40 machines from VGrADS testbed for 14 days 7 hours and 44 minutes

Table 2. Batch jobs used to solve the k2fix-gr-rcs-w9.cnf instance from SAT 2003 benchmark

In order to illustrate further GridSAT's success in using all the above variety of resources mentioned earlier we present a section of a run using instance *hanoi6.cnf*. This problem is a SAT representation of the *Hanoi Towers* problem using six disks. A six day snapshot from a 23 day run is shown in figure 3(a) using logarithmic scale. The figure shows several jobs from Blue Horizon, Data Star and TeraGrid sites participating in the execution. This figure shows that GridSAT was able to make use of the available resource when some of their nodes became available and then continued to run after the nodes were taken away to serve other users. GridSAT processes continue to run on the batch controlled resources until the scheduler decides to terminate them. This abrupt termination has no effect on the application which deals with these events as (scheduled) resource failures. GridSAT was able to manage up to 350 processes running on different resources as show in this figure.

The satisfiability solver performs mostly integer, branching and load/store operations. The number of floating point operations is very low (less than .1 FLOPS). We present in figure 3(b) an estimate of the total number of instructions per second during the same six day period. Since instrumenting GridSAT can cause significant slow down, we conducted some benchmarking on some machines at UTK to determine the average efficiency of the solver. Since the solver code is mostly sequential, we assume that at the maximum only one instruction per cycle can be finished by the processor. The determined efficiency is 70%. We estimated that other hardware and OS combinations will exhibit equal efficiencies. The number of operations provided by a resource is estimated to be the product of its peak performance and the estimated efficiency. The total number of instructions in figure 3(b) is the sum of operations of all active resources. We notice that the VGrADS testbed is able to deliver about 20 Billion instructions per second (IPS). In the middle of the graph, there is a batch job from Blue Horizon which failed suddenly while joining the GridSAT execution. This might have happened because the Blue Horizon machine became unavailable for scheduled maintenance. The total number of IPS was multiplied by more than five times when some batch jobs became active. It reached up to 110 Billion IPS.

Another measure of performance, is how much of the batch job maximum computational power is actually used by GridSAT processes. Most other parallel jobs run on all the processes from start to finish with little overhead. In this case, batch jobs are efficiently used. In the of case GridSAT, however, there are two main sources of inefficiency. First, some jobs might wait ideally at the start. Batch jobs usually include a large number of processes. Some of these processes have to wait until a sufficient number of splits occur to generate new sub-problems for all the newly created solvers. Second, some batch processes may contain idle solvers for a period of time after they solve the previously assigned sub-problem. The solver in this case, waits until it is assigned a new sub-problem by the master. For the first job in figure 3(a), which is a large 100-node job, the efficiency is 98.9%. Thus GridSAT was able to use batch jobs efficiently. The main reason is that batch jobs usually wait in the batch queue for a long time before executing. Thus by the time the job is executed, GridSAT was unable to solve the problem because it is hard. This means that batch jobs are only used when the problem is in deed hard. It is possible that for certain problems, the efficiency of batch jobs might be low. In this case, future versions of GridSAT might monitor the batch job efficiency to determine whether and when a job is to be terminated.

Compute resource	node- -hours	CPUs/ node	CPU -hours
BlueHorizon	3200	8	25600
DataStar	160	11	1760
TG@SDSC	1080	2	2160
TG@NCSA	200	2	400
GrADS(*)	13750	1	13750

(*) machines were shared with other users

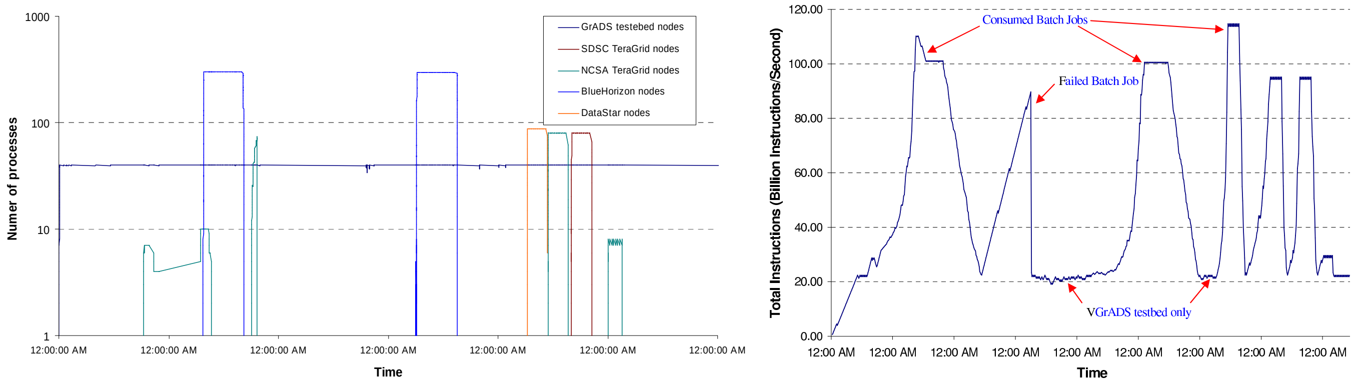
Table 3. CPU-hours per resource used to solve the k2fix-gr-rcs-w9.cnf instance from SAT 2003 benchmark

During our experiments, the Blue Horizon super-computer was being decommissioned. GridSAT was able to continue running experiments on the set of available resources through this transition. The scheduler would try to submit jobs but it would notice that the Blue Horizon resource was not responding. The failure of this single (but important) resource which did not affect the already running experiments shows the robustness of GridSAT.

6 Conclusion

This paper presents a new version of GridSAT which implements a parallel, distributed and complete satisfiability solver. In order to solve harder problems, new improvements to both the algorithm and architecture of GridSAT were introduced. GridSAT is capable to dynamically selecting resources to enable improved overall performance.

The experiments we presented show GridSAT's ability to manage and use a diverse set of dynamic computational grid resources. The experiments lasted for weeks as a testament to the robustness of the application. During these experiments new previously unsolved problems from practical and theoretical fields were solved.



(a) Processor Count (b) Instructions Per Second

Figure 3. A six day snapshot representing (a) processor count usage in logarithmic scale and (b) estimated instructions per second (IPS) usage for all resources during a six day period.

References

[1] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf. The Cactus Worm: Experiments with dynamic resource discovery and allocation in a Grid environment. *The International Journal of High Performance Computing Applications*, 15(4):345–358, 2001.

[2] 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. *International Journal of High Performance Computing Applications*, 15(4), Winter 2001. available from "`http://hipersoft.cs.rice.edu/grads/publications_reports.htm`".

[3] A. Biere. Limmat. `http://www.inf.ethz.ch/personal/biere/projects/limmat/`.

[4] W. Chrabakh and R. Wolski. GrADSAT: A Parallel SAT Solver for the Grid. Technical Report 2003-05, UCSB, March 2003.

[5] W. Chrabakh and R. Wolski. GridSAT: A chaff-based Distributed SAT solver for the Grid. In *Supercomputing Conference, Phoenix, AZ*, November 2003.

[6] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. In *Proc. 10th IEEE Symp. on High Performance Distributed Computing*, 2001.

[7] M. Davis, G. Logeman, and D. Loveland. A machine program for theory proving. Communications of the ACM, 1962.

[8] S. L. Forman and A. M. Segre. Nagsat: A randomized, complete, parallel solver for 3-sat. SAT2002, 2002.

[9] M. P. I. Forum. Mpi: A message-passing interface standard. Technical Report CS-94-230, University of Tennessee, Knoxville, 1994.

[10] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 1997.

[11] I. Foster and C. Kesselman, editors. *The Grid – Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[12] E. Goldberg and Y. Novikov. BerkMin: A fast and robust SAT-solver. In *Design, Automation, and Test in Europe (DATE '02)*, pages 142–149, March 2002.

[13] GrADS. `http://hipersoft.cs.rice.edu/grads`.

[14] S. Graham and B. Murray. http://docs.oasis-open.org/wsn/2004/06/wsn-WS-BaseNotifi cation-1.2-draft-03.pdf.

[15] E. A. Hirsch and A. Kojevnikov. UnitWalk: A new SAT solver that uses local search guided by unit clause elimination. In *PDMI preprint 9/2001, Steklov Institute of Mathematics at St.Petersburg*, 2001.

[16] D. Jackson and M. Vaziri. Finding bugs with a constraint solver. *International Symposium on Software Testing and Analysis*, 2000.

[17] B. Jurkowiak, C. M. Li, and G. Utard. Parallelizing Satz Using Dynamic Workload Balancing. In *Proceedings of Workshop on Theory and Applications of Satisfi ability Testing (SAT'2001)*, pages 205–211, June 2001.

[18] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks: Logic Synthesis and Verifi cation Using Techniques*. Kluwer Academic Publishers, Boston, 1997.

[19] T. Larrabee. Effi cient generation of test patterns using boolean difference. pages 795–802.

[20] T. Larrabee. Test pattern generation using boolean satisfi ability. In *IEEE Transactions on Computer-Aided Design*, pages 4–15, January 1992.

[21] W. W. Li, R. W. Byrnes, J. Hayes, A. Birnbaum, V. M. Reyes, A. Shahab, C. Mosley, D. Pekurovsky, G. B. Quinn, I. N. Shindyalov, H. Casanova, L. Ang, F. Berman, P. W. Arzberger, M. A. Miller, and P. E. Bourne. The encyclopedia of life project: grid software and deployment. *New Gen. Comput.*, 22(2):127–136, 2004.

[22] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. "Chaff: Engineering an Effi cient SAT Solver. 38th Design Automation Conference (DAC2001), Las Vegas, June 2001.

[23] G. Nam, F. Aloul, K. Sakallah, and R. Rutenbar. A Comparative Study of Two Boolean Formulations of FPGA Detailed Routing Constraints. *International Symposium on Physical Design (ISPD), Sonoma Wine County, California*, pages 222–227, 2001.

[24] OASIS Web Services Reosurce Framework (WSRF) TC. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`.

[25] SAT 2002 Competition. http://www.satlive.org/satcompetition/.

[26] SAT 2003 Competition. `http://satlive.org/SATCompetition/2003/`.

[27] M. H. Schulz and E. Auth. Improved Deterministic Test Pattern Generation with Applications to Redundancy Identifi cation. *IEEE Transactions on ComputerAided Design*, 8(7):811816, July 1989.

[28] J. M. Silva and K. Sakallah. Grasp - a new search algorithm for satisfi ability. ICCAD. IEEE Computer Society Press, 1996.

[29] J. P. M. Silva. Search Algorithms for Satisfi ability Problems in Combinational Switching Circuits. Ph.D. Thesis, The University of Michigan, 1995.

[30] C. Sinz, W. Blochinger, and W. Kuchlin. PaSAT - Parallel SAT-Checking with Lemma Exchange: Implementation and Applications. In *Proceedings of SAT2001*, pages 212–217, 2001.

[31] T. Tannenbaum and M. Litzkow. The condor distributed processing system. *Dr. Dobbs Journal*, February 1995.

[32] TeraGrid. http://www.teragrid.org/.

[33] The W3C Web Services Architecture working group, public draft, August 2003. `http://www.w3.org/TR/2003/WD-ws-arch-20030808/`.

[34] S. Tuecke, L. Liu, and S. Meder. http://docs.oasis-open.org/wsrf/2005/03/wsrf-WS-BaseFaults-1.2-draft-04.pdf.

[35] VGrADS. `http://hipersoft.cs.rice.edu/vgrads`.

[36] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su. Running EveryWare on the Computational Grid. In *Proceedings of SC99,*, November 1999.

[37] R. Wolski, N. Spring, and J. Hayes. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 1999.

Grid computing for energy exploration

Dimitri Bevc[1], Sergio E. Zarantonello[2], Neena Kaushik[3], Iulian Musat[4]

Abstract

3-D seismic imaging is the most computationally intensive task in the oil and gas industry. It is a key technology that has allowed the success ratio of exploratory wells to increase from 20% to 80%. The objective of this paper is to give an overview of a Grid-enabled environment for seismic imaging developed by 3DGeo and comment on its use in a production setting. This system addresses the demand for advanced seismic imaging applications in the oil and gas industry, and the ensuing need of computational and data resources to run these applications flexibly and efficiently.

1. Introduction

A key task in exploration geophysics is the creation of images of the subsurface of the earth to identify deposits of oil and gas. The earth's response to trains of artificially created elastic waves is recorded with arrays of geophones. This data is then processed to ultimately deliver images of the subsurface. The construction of accurate 3-D images of the subsurface from this data is an extremely resource-intensive task. It requires the handling of large data volumes - on the order of 10-15 Terabytes for a single modern marine 3-D survey - and the application of computationally-demanding imaging algorithms. Only large-scale parallel computers can apply these algorithms and deliver the results within a useful turn-around time. Furthermore, the most advanced imaging technologies are resource-demanding and only feasible today on small-sized projects. Harnessing Grid resources flexibly and effectively is thus a fundamentally important development for the oil and gas industry.

In this paper we describe our work developing a Grid-enabled system for seismic imaging.

3DGeo is a leading-edge provider of advanced software products and services to the oil and gas industry[5] . Its commercial offerings include INSP (Bevc and Popovici, 2003), a proprietary Java based Internet infrastructure for remote collaborative seismic processing

[1] 3DGeo Development Inc., 4633 Old Ironsides Drive, Santa Clara, CA 95054 (**dimitri@3dgeo.com**).
[2] 3DGeo Development, Inc., 4633 Old Ironsides Drive, Santa Clara, CA 95054 (**sergio@3dgeo.com**), and Department of Applied Mathematics, Santa Clara University, 500 El Camino Real, Santa Clara, CA 95053 (**szarantonello@scu.edu**).
[3] 3DGeo Development, Inc., 4633 Old Ironsides Drive, Santa Clara, CA 95054 (**neena@3dgeo.com**), and Department of Computer Engineering Santa Clara University, 500 El Camino Real, Santa Clara, CA 95053 (**nrkaushik@scu.edu**).
[4] 3DGeo Development Inc., 17171 Park Row, Houston, TX 77084 (**iulian@3dgeo.com**).
[5] 3DGeo Development, Inc., **http://www.3dgeo.com/**

and seismic imaging, and a suite of advanced imaging applications that can be accessed, executed, and monitored with the INSP system. The conversion of INSP to a Grid-enabled system, providing flexible and secure access to advanced imaging applications and to the resources to run these applications *whenever and wherever needed,* is a strategic step for 3DGeo. We believe such a system will allow 3-D seismic data to be much more effectively used to characterize and delineate oil reservoirs and contribute to opening up exploration venues in extremely complicated geological conditions. A major oil discovery in these areas could decrease United State's dependence on imported oil and have a direct impact on the cost of energy.

The organization of the paper is as follows. In Section 2 we give an overview of 3-D depth imaging, we discuss the computational cost of different algorithms, and give an overview of Kirchhoff depth migration, a typical 3-D imaging application. In Section 3 we discuss design issues associated to PSDM, a particular implementation of Kirchhoff migration, in a Grid environment. In Section 4 we give an overview of the INSP seismic processing system and the specifics of the Grid-enabled extension. In Section 5 we discuss the implications of seismic imaging on the Grid and give our conclusions.

2. Seismic imaging

Seismic imaging methods solve an inverse problem for the classical wave equation of mathematical physics, whereby signals recorded by geophones on the surface of the Earth are converted to a model of the subsurface. 3-D depth imaging methods are the most computationally-intensive tasks in the oil and gas industry. These methods are usually classified into two categories: methods based on the Kirchhoff integral equation, and methods that operate directly with the wave equation. Wave-equation methods are further classified as shot-receiver (*e.g.* common azimuth and narrow azimuth methods) and shot-profile methods. Wave equation shot profile methods are the most computationally expensive methods in use, and are unfeasible except for small projects.

Table 1. The computational challenge: Gulf of Mexico 3-D marine surveys.

Size of data		***Runtime in days***		
Blocks	***Gbytes***	***Kirchhoff***	***Narrow azimuth***	***Shot profile***
10	620	3	31	184
100	6,200	111	1,100	6,640
500	30,700	996	9,960	59,800 (164 yrs !)

To put the computational challenge in perspective, in Table 1 we compare the estimated runtimes of hypothetical imaging projects for Deep Gulf of Mexico 3-D marine surveys on a 128-CPU cluster of 2.4 GHz Pentium® 4 processors delivering a sustained performance of 900 Mflops/CPU. Running shot profile migration on a large 3-D marine survey would take 164 years to complete on such system ! The INSP system has modules for all the methods represented in Table 1.

The better accuracy offered by advanced methods is illustrated in Figure 1, where we compare the seismic images obtained with PSDM, 3DGeo's implementation of Kirchhoff pre-stack 3-D depth migration, and the image obtained with one of 3DGeo's wave-equation migration methods (Biondi and Palacharla, 1996). The generally computationally more intensive wave-equation method gives better accuracy than Kirchhoff migration, therefore underscoring the need for more compute resources deliverable through the Grid.

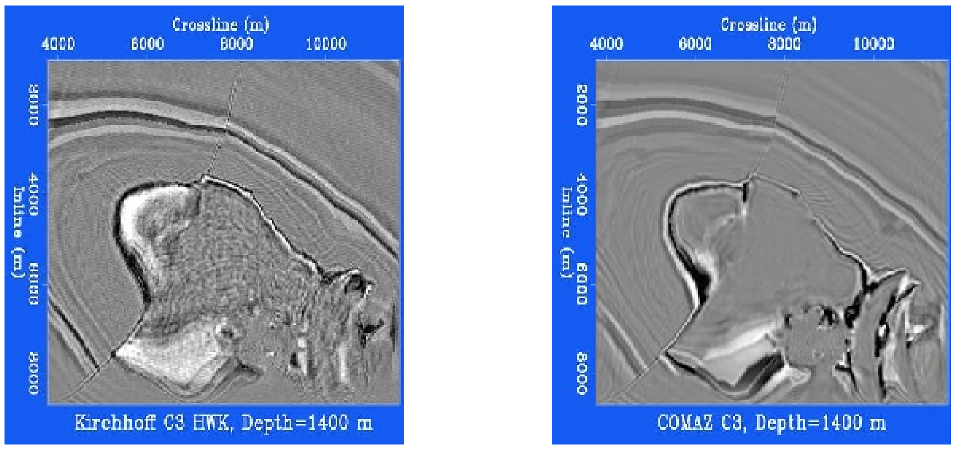


Figure 1. Image on left obtained using PSDM. Image on right obtained using a wave-equation migration method.

3. Overview of PSDM

We use PSDM as an example to illustrate design issues that were addressed for Grid deployment. Pre-Stack Depth Migration (PSDM) is 3DGeo's implementation of the three-dimensional Kirchhoff depth migration, one of the most comprehensive and flexible methods for imaging pre-stack 3-D seismic data. PSDM approximately solves the wave equation with a boundary integral method. The acoustic reflectivity at every point of the Earth's interior is computed by summing the recorded data on multidimensional surfaces; the shape of the summation surfaces and the summation weights are computed from the Green's functions of the single scattering wave propagation experiment.

The essence of 3-D prestack Kirchhoff migration can be expressed in the following integral equation:

$$\mathbf{Image}(\mathbf{x}) = \int \int_{\mathbf{x_s}} \int_{\mathbf{x_r}} \mathbf{G}(\mathbf{x_s}, \mathbf{x}, \omega)\, \mathbf{G}(\mathbf{x}, \mathbf{x_r}, \omega)\, \mathbf{Data}(\mathbf{x_s}, \mathbf{x_r}, \omega)\, \mathbf{dx_r}\, \mathbf{dx_s}\, \mathbf{d}\omega$$

If the Green's functions are known this solution is exact. In a computational environment we express the integral as a sum:

$$\mathbf{Image}(\mathbf{x}) = \sum_{\mathbf{x_s}} \sum_{\mathbf{x_r}} \mathbf{A_s A_r}\, \mathbf{Input}(\mathbf{x_s}, \mathbf{x_r}, \mathbf{t_s} + \mathbf{t_r})$$

where A_r and A_s are determined by the transport equation, and t_r and t_s are either found by ray-tracing or by solving the eikonal equation. We note that each point **x** of the seismic image is calculated by a sum over a travel-time surface, and that the sums for the different points of the image can be calculated independently from each other. Since the wave propagation velocity is not known *a priori*, the process of building an exact image is iterative, with successive improvements made to the velocity field. Each iteration requires running a Kirchhoff migration. The overall procedure, schematized in Figure 2, involves collaboration between multidisciplinary teams and can be extremely demanding in terms of human and computational resources.

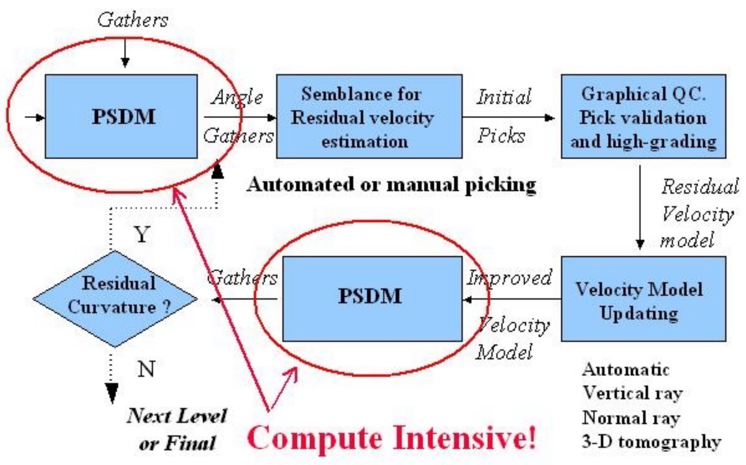


Figure 2. Building a 3-D seismic image requires iterative updating of the velocity model.

Parallelization of PSDM on a cluster. PSDM presents different parallelization issues in cluster and Grid environments. 3DGeo's implementation of Kirchhoff migration was designed to achieve maximum efficiency on a cluster of interconnected multiprocessor computers. To achieve this efficiency the input data is distributed among the computational nodes, while the output image is divided into processing blocks that are distributed over parallel processors on each node. At the end, the results from each node are gathered to build the final image.

Figure 3 illustrates the PSDM MPI architecture. The input data is distributed using a dispatching service which responds to individual requests from each node. This approach ensures a good load balancing over heterogeneous nodes, especially if the computational demands for processing each block of input data are different. This dispatching service keeps track of the execution stage and provides necessary information to restart the entire job in case of a failure, an important feature on a large distributed system where one node may become unavailable anytime during the execution.

Each node has a copy of the output image which is divided into processing blocks. The summation computation for each block is distributed independently over parallel processors. By partitioning the output, the algorithm is scalable with respect to the total amount of memory available, and can run efficiently on workstations where less memory is available or on supercomputers where any remaining memory can be used for staging data. Once the entire input data is processed, the final image is composed by summing the local images from each node.

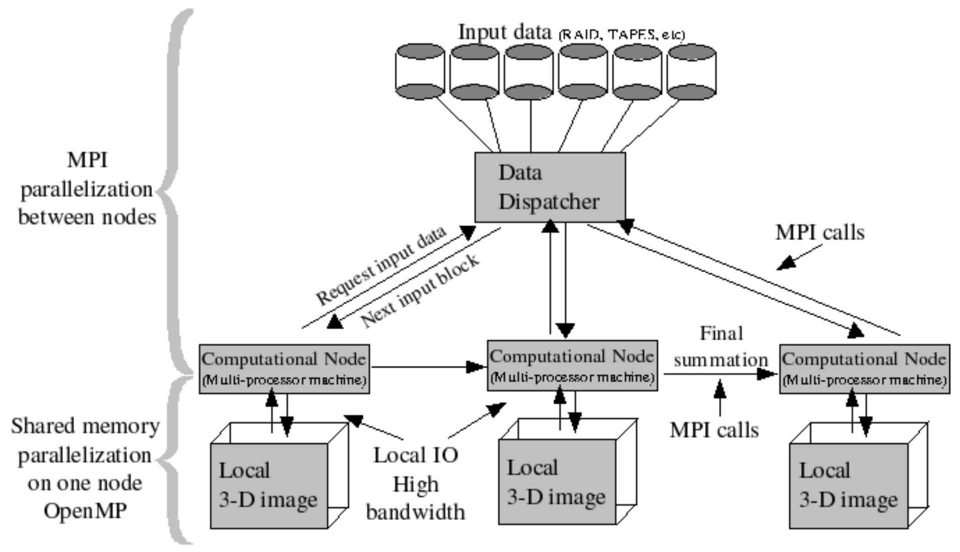

Input data (RAID, TAPES, etc)
MPI parallelization between nodes
Data Dispatcher
Request input data
Next input block
MPI calls
Final summation
Computational Node (Multi-processor machine)
Computational Node (Multi-processor machine)
Computational Node (Multi-processor machine)
MPI calls
Shared memory parallelization on one node OpenMP
Local 3-D image
Local IO High bandwidth
Local 3-D image
Local 3-D image

Figure 3. Parallelization of PSDM on a cluster

After comparing several multi-threading libraries, we chose the OpenMP[6] standard for the implementation of the shared-memory parallelization on each multiprocessor node. This standard is jointly defined by a group of major computer hardware and software vendors, and provides a simple and flexible interface for developing parallel applications, portable on a variety of systems.

The solution chosen to implement the input data dispatching mechanism and the collective operations used for the final summation is based on the MPI standard. We used MPICH (Groop, Doss, and Skjellum, 1996), an open source implementation of the MPI standard, optimized and tested on a variety of architectures. PSDM uses about one hundred parameters, usually extracted from a text file. Since many of those parameters refer to files on the local file system, the parameter acquisition library was augmented to accept placeholders for different MPI runtime values such as the process rank. This simplifies the laborious task of setting up a PSDM running job.

PSDM on the Grid. Tests and benchmarking of PSDM on various cluster architectures and configurations shows that in a typical run the I/O operations associated with the input data distribution account for a small fraction of the total processing time. This encouraged us to use - in a first phase - the same architecture for distributing a PSDM job across multiple clusters, interconnected in a computational grid, as shown in Figure 4. Using the Globus Toolkit (Foster and Kesselman, 2003) we set up a Grid, interconnecting two of 3DGeo's processing centers (Santa Clara, California, and Houston, Texas) and a Linux cluster at the San Diego Supercomputing Center.

We had built PSDM using MPICH libraries, and the choice of MPICH-G2 (Karonis, Toonen, and Foster, 2003) for the necessary Grid support seemed a natural decision. The result was a Grid-enabled MPI implementation built on top of MPICH and Globus API. After re-compiling with the MPICH-G2 libraries, we were able to deploy and run a PSDM job on public IP nodes. However, since MPICH-G2 does not support private IP clusters, running a single PSDM job on two or more interconnected clusters was not possible and a modified approach was required.

The main problem was due to the implementation of MPICH-G2 which required direct TCP/IP connection between the nodes of the different clusters. Since most of the clusters were configured without public IP addresses for the internal nodes, direct TCP/IP connection could not be established. For testing purposes we were able to tunnel the TCP traffic through a secure connection, setting up a VPN between two test clusters. This solution was not scalable and the settings are impractical to achieve on a commercial Grid setup where different participants have different security policies. However, the test was fruitful in assessing the limitations of the parallelization model we built for PSDM and to draw future development plans for improving it. Our present plans are to explore and use the functionality of MPICH-GP, a modification of MPICH-G2 by Kum-Rye Park. MPICH-GP includes a Network Address Translation service, and a user-level proxy scheme. MPICH-GP supports public as well as private IP clusters, and mixed combinations of public and private IP clusters (Park *et al.*, 2004).

[6] http://www.openmp.org/

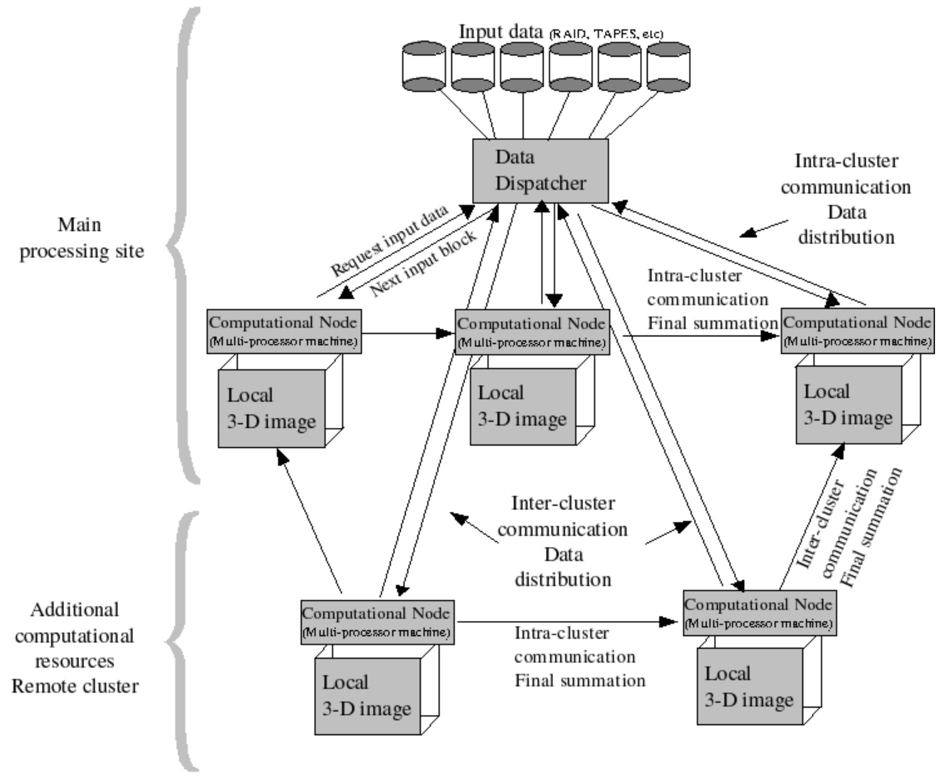

Input data (RAID, TAPES, etc)
Data Dispatcher
Main processing site
Request input data
Next input block
Intra-cluster communication Data distribution
Intra-cluster communication Final summation
Computational Node (Multi-processor machine)
Local 3-D image
Inter-cluster communication Data distribution
Inter-cluster communication Final summation
Additional computational resources Remote cluster

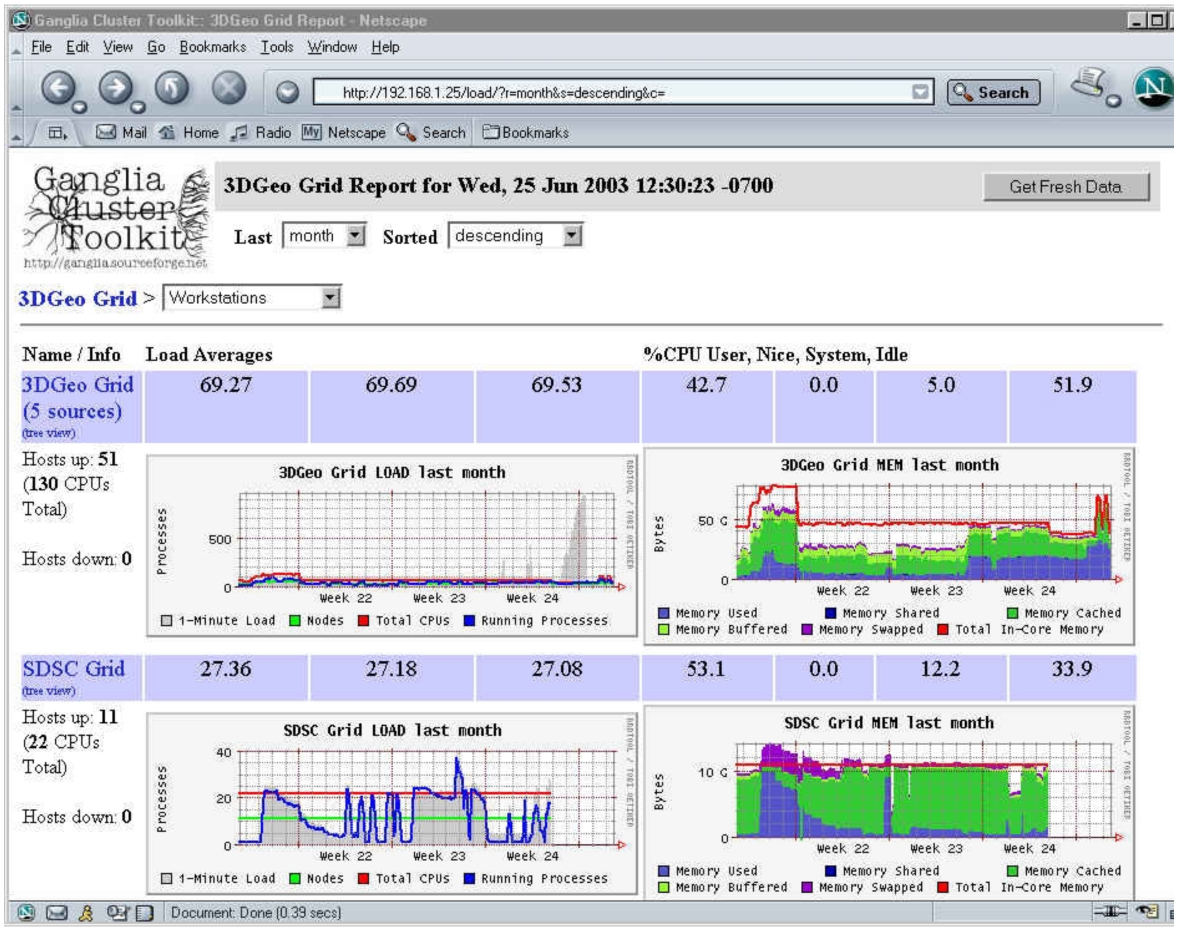
Figure 4. Parallelization of PSDM on multiple clusters.

Demonstration on the Virtual Computer. Once the SDSC cluster was configured to support the seismic imaging software, it was connected to 3DGeo's distributed monitoring Grid. The connection to the monitoring grid was performed by installing the Ganglia Resource Monitoring tool. Ganglia[7] is an Open Source application, originally developed by the Berkeley Millenium project and currently bundled with the NPACI Rock distributions. Ganglia was used to interconnect the 3DGeo Santa Clara processing center, consisting of 32 CPUs and 80 CPUs Linux clusters, with 3DGeo's Houston processing center, 5 clusters hosting another 350 CPUs Linux cluster. The SDSC cluster was the last addition to the monitoring grid. Ganglia gave an overview of resource utilization.

The information is accessible as a web page (Figure 5), and includes graphs showing the evolution in time of metrics such as machine load, memory usage, and number of processes. 3DGeo processing staff were thus able to inspect from a single web page the load of the machines geographically distributed in Santa Clara CA, San Diego CA and Houston TX.

[7] http://ganglia.sourceforge.net/

From the system administration point of view, setting up the monitoring infrastructure consisted of the installation and configuration of the Ganglia monitor daemon on all the cluster nodes, including the master nodes, and the installation of a Ganglia metadaemon, an Apache web server, and the Ganglia web front end on the master nodes. The central metadaemon running in the Santa Clara processing center was configured to interrogate and accept data from the San Diego cluster metadaemon. In addition, Gexec, a component of the Ganglia project, was experimentally installed on 3DGeo's and SDSC's clusters. Gexec is a scalable cluster remote execution system which provides RSA authenticated remote execution of parallel and distributed jobs. Its capabilities include transparent forwarding of stdin, stdout, stderr, and signals to and from remote processes, it provides local environment propagation. Gexec is designed to be robust and to scale to systems of over 1000 nodes. For our experimental grid we installed Globus server software GT 2.4 on the master nodes of all the clusters. The final step was to incorporate the Grid-enabled applications such as PSDM within the INSP framework.


Ganglia Cluster Toolkit: 3DGeo Grid Report - Netscape
File Edit View Go Bookmarks Tools Window Help
http://192.168.1.25/load/?r=month&s=descending&c=
Search
Mail Home Radio Netscape Search Bookmarks
Ganglia Cluster Toolkit
http://ganglia.sourceforge.net
Get Fresh Data
3DGeo Grid LOAD last month
Processes
500
0
Week 22 Week 23 Week 24
☐ 1-Minute Load 🟩 Nodes 🟥 Total CPUs 🟦 Running Processes
3DGeo Grid MEM last month
Bytes
50 G
0
Week 22 Week 23 Week 24
🟦 Memory Used 🟦 Memory Shared 🟩 Memory Cached
🟩 Memory Buffered 🟪 Memory Swapped 🟥 Total In-Core Memory
SDSC Grid LOAD last month
40
20
0
SDSC Grid MEM last month
10 G
RRDTOOL / TOBI OETIKER
Document: Done (0.39 secs)

3DGeo Grid Report for Wed, 25 Jun 2003 12:30:23 -0700

Last month Sorted descending

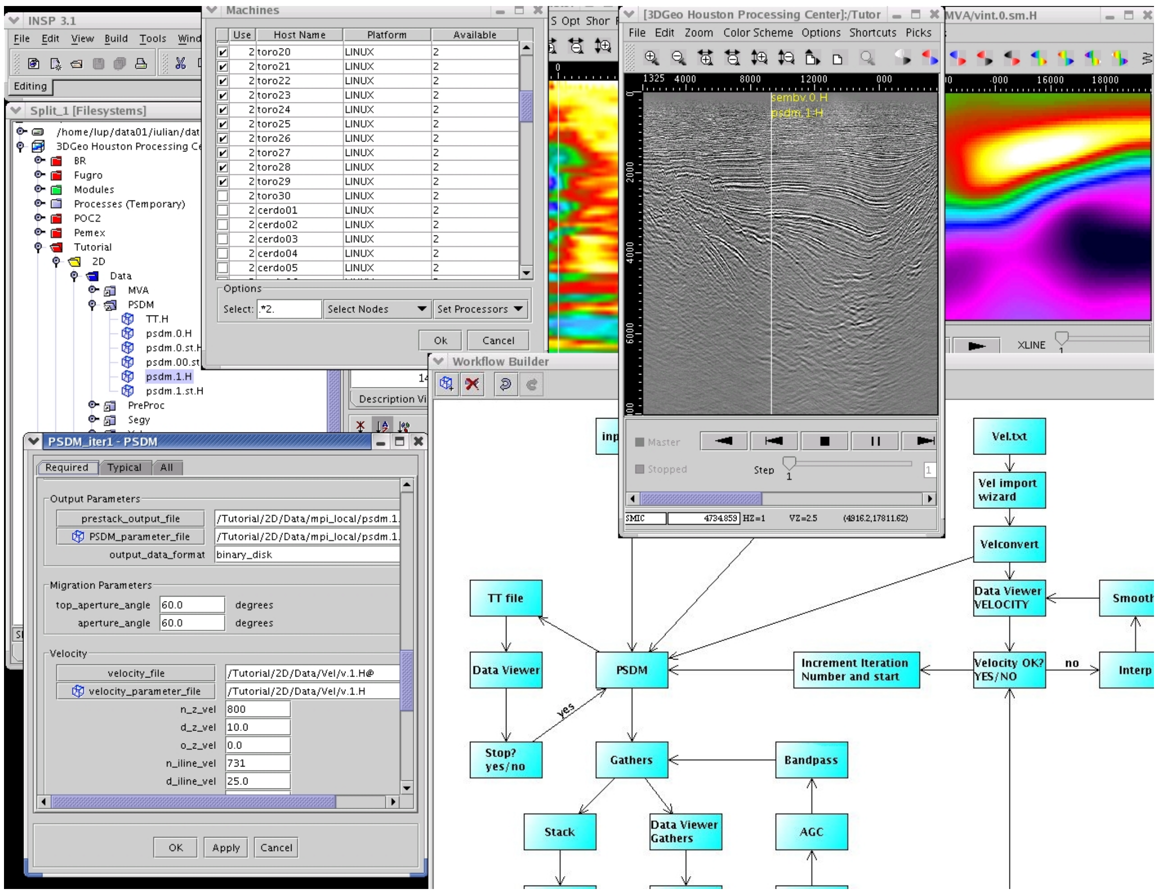
3DGeo Grid > Workstations

Name / Info	Load Averages			%CPU User, Nice, System, Idle			
3DGeo Grid (5 sources) (tree view)	69.27	69.69	69.53	42.7	0.0	5.0	51.9
Hosts up: 51 (130 CPUs Total) Hosts down: 0							
SDSC Grid (tree view)	27.36	27.18	27.08	53.1	0.0	12.2	33.9
Hosts up: 11 (22 CPUs Total) Hosts down: 0							

Figure 5. Status of the 3DGeo Grid displayed by Ganglia in Web browser. The windows display CPU and memory load of the available 3DGeo and SDSC resources.

4. Tying Grid resources together with INSP

INSP is a collaborative environment, developed and commercialized by 3DGeo for building and launching workflows of computationally intensive parallel and distributed jobs, visualizing data on client workstations, and monitoring jobs. It is based on a client-server relationship between user interfaces (clients) and computationally intensive workflows (servers), presenting users with options to visualize intermediate results, and monitor, and design workflows for seismic processing (Bevc, Popopvici, and Biondi, 2002). The INSP Data Viewer allows a user to visualize a data cube, to pick locations, and to edit values in the data cube, thereby developing a geological earth model. It allows remote clients to interact with multiple data objects. Figure 6 shows a screen shot of the INSP Explorer interface illustrating some of the INSP functionalities. Visible in the left side is the explorer tree structure of modules (executable application-specific programs), datasets, and workflows for each server. Seismic data, velocity models, and a sample workflow are displayed on superimposed windows.



	Use	Host Name	Platform	Available
✓	2	toro20	LINUX	2
✓	2	toro21	LINUX	2
✓	2	toro22	LINUX	2
✓	2	toro23	LINUX	2
✓	2	toro24	LINUX	2
✓	2	toro25	LINUX	2
✓	2	toro26	LINUX	2
✓	2	toro27	LINUX	2
✓	2	toro28	LINUX	2
✓	2	toro29	LINUX	2
	2	toro30	LINUX	2
	2	cerdo01	LINUX	2
	2	cerdo02	LINUX	2
	2	cerdo03	LINUX	2
	2	cerdo04	LINUX	2
	2	cerdo05	LINUX	2

Options
Select: .*2.

PSDM_iter1 - PSDM

Output Parameters
prestack_output_file /Tutorial/2D/Data/mpi_local/psdm.1.
PSDM_parameter_file /Tutorial/2D/Data/mpi_local/psdm.1.
output_data_format binary_disk

Migration Parameters
top_aperture_angle 60.0 degrees
aperture_angle 60.0 degrees

Velocity
velocity_file /Tutorial/2D/Data/Vel/v.1.H@
velocity_parameter_file /Tutorial/2D/Data/Vel/v.1.H
n_z_vel 800
d_z_vel 10.0
o_z_vel 0.0
n_iline_vel 731
d_iline_vel 25.0

Figure 6. INSP Explorer is the Internet-based GUI for remote processing services.

The INSP graphical user interface presents the users with a complete set of seismic processing and imaging modules and a visualization system. The GUI is written in Java, allowing client portability and access to any type of computer on either a local or wide-area network. This takes advantage that Java can handle security and parallel distributed computing, all key issues in geophysical applications. The Java client-server design of INSP leverages the "write once, run anywhere" capabilities for the GUI and process management, while using highly optimized seismic imaging algorithms running on specialized high performance computers for number-crunching tasks. A key function of INSP is to facilitate the communication between members of a geographically distributed exploration team, providing them with the tools that help them easily share information, regardless of physical distance between members of the team.

Design of current INSP. The current INSP consists of a 3dGeo proprietary client and server. The client part is the GUI which shows the data, flows and the modules. It also allows the user to mount remote file systems, view data, and add new workflows. The modules are populated in the database on the server side using the "inspac" command. They are viewable on the client GUI when connected to the INSP server. The INSP architecture is shown below.

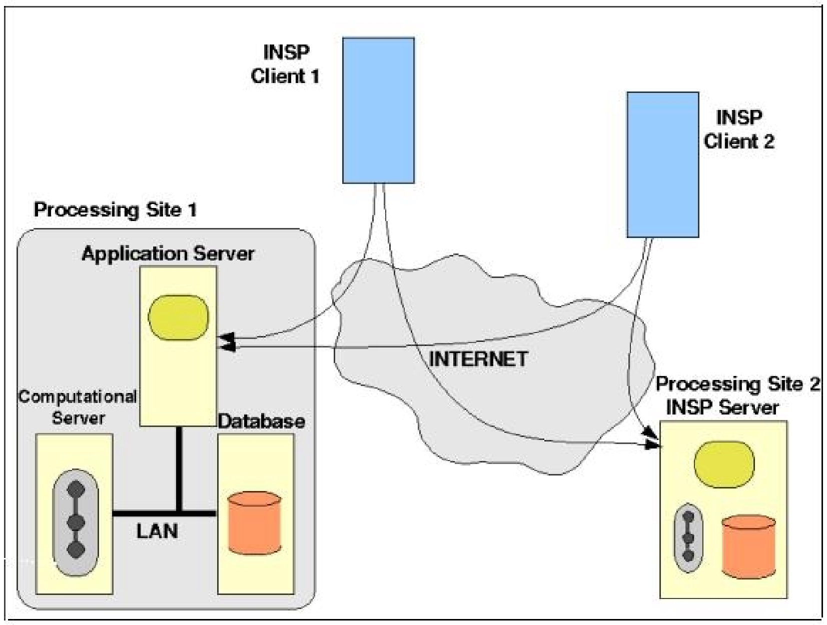
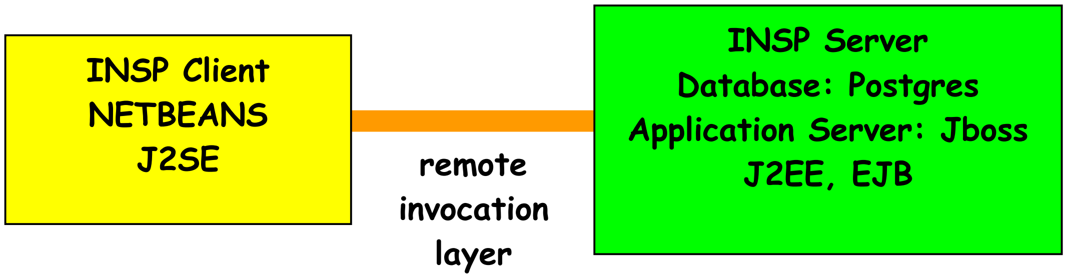
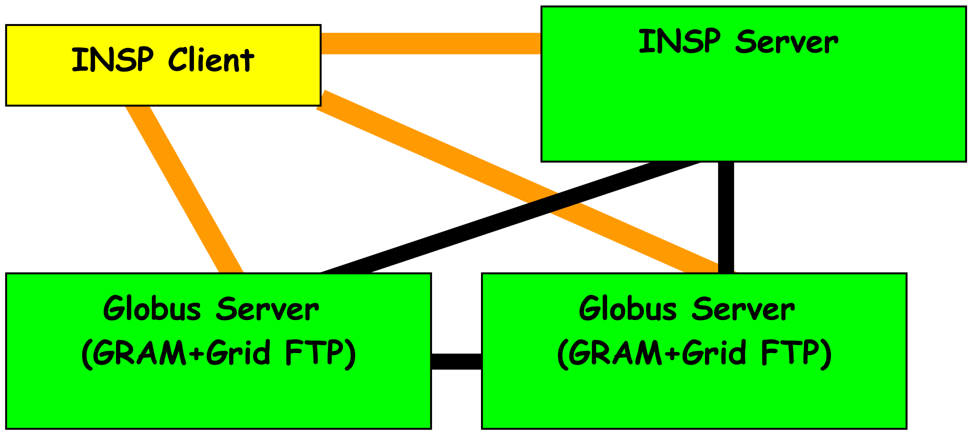


Figure 7. Current INSP architecture.

Grid-enabled INSP. We are currently redesigning INSP to allow the client to use the Globus server to run flows and view/manage data. The objective is to replace the current INSP design:



with a Grid-enabled design shown below:



In this design, security will be handled by the Globus server. The A&A policies will become Grid-specific. While the current INSP server provides an additional layer of authorization (ACL type of permissions for every project), the same can be done at the filesystem level, so any system administrator will be able to handle this task without having to learn a new system. The flows will be simple files which can reside either on the local disk or on the server. So the sharing will be done in the same way as with data files – the user can run his/her own flows or can save them on a server for later use. When connecting from different machines to the same server, the same list of files will be seen, the same as with an FTP client. The flows and data will be logically separated on what we call the project database, which will be an XML file containing all necessary information for a seismic project plus the paths to data and flows. In this framework it is possible to have exactly the same structure as in the present version of INSP - and this is the objective – to enhance it for better usability in a Grid environment. Our plans are to incorporate Ganglia in INSP by creating a Java API to Ganglia: The Ganglia monitor daemon collect cpu load data, etc. and broadcasts this info.The Ganglia metadaemon runs on a single node and collects this info in an XML database. Our objective is to have an API on Java that can access the monitor daemon.

5. Conclusions

Seismic imaging is an application area where Grid computing holds great promise. Today's operational environment, shown in Figure 8, involves many inefficiencies that are seamlessly resolved in a Grid environment. The data tapes are transported physically (typically by UPS or FedEx) between the raw data acquisition site, data banks, data processing sites (usually a seismic contactor), and quality control and interpretation sites - the customer oil and gas production company. Once the data is interpreted it is often reprocessed with a new set of parameters. The process is repetitious and lengthy, culminating in decision to drill a well which can cost in the order of $30 M. In today's operational environment, a large imaging project can easily require one to two years to complete.

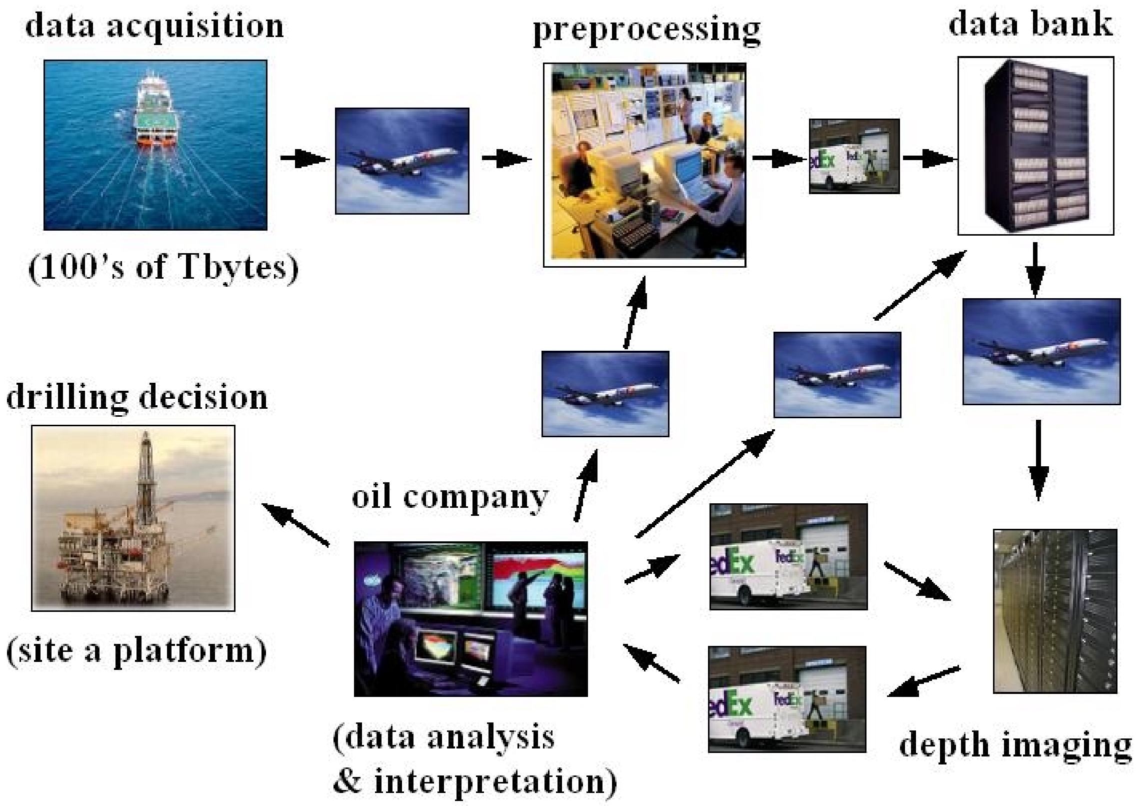


Figure 8. In today's operational scenario data are physically transported between the acquisition, processing, and storage sites, and the end user (oil company).

In a fully-optimized Grid operational environment, shown in Figure 9, we expect the time to complete the same project to be reduced to six months or less. This framework will necessitate a wavelet-based compression utility to allow for electronic transmission of extremely large datasets (Bevc *et al.*, 2004). The data will be monitored as it is being acquired, quality control will be concurrent with data processing, and interpretation and reprocessing will be done with much greater flexibility. These advantages are compelling, and have motivated 3DGeo's efforts to be at the forefront of bringing the Grid to the energy exploration industry.

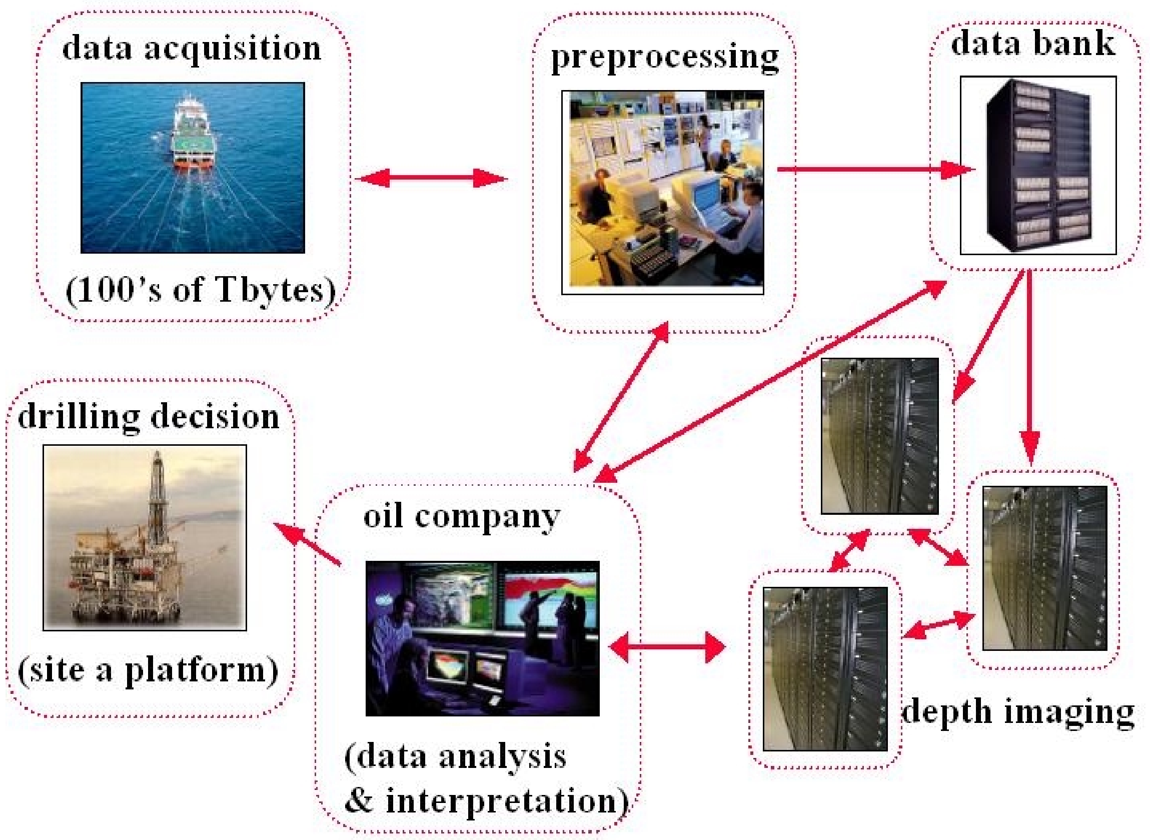


Figure 9. The future : a fully-enabled Grid scenario that by allowing greater and more flexible access to resources (data, computers, personnel), reduces turn around time and dramatically shortens the time to making a drilling decision. In this scenario, all components of the process become Grid nodes accessible through INSP. Data transfers are via GridFTP; resource requests are through Globus Resource Allocation Manager.

A Grid-enabled environment for seismic imaging allows for a new paradigm of commodity computing. In the life of a seismic imaging project the demand for compute cycles will ebb and swell as the project proceeds through its different stages. Buying compute cycles on the Internet whenever needed will free the oil and gas exploration industry to focus on its core competencies, and will result in dramatically increased productivity.

References

Bevc, D., Donoho, D. L., and Zarantonello, S. E., 2004, "Application of 2nd generation wavelets in seismic imaging", *Proceedings of 74'th Ann. Internat. Mtg: Soc. of Expl. Geophys.*

Bevc, D., and Popovici, A., M., 2003, "Integrated Internet collaboration", *The Leading Edge*, **22**, pp. 54-57.

Bevc, D., Popovici, M., and Biondi, B., 2002, "Will Internet processing be the new paradigm for depth migration interpretation and visualization ?", *First Break*, **20**, no. 3.

Biondi, B., and Palacharla, G., 1996, "3-D prestack migration of common-azimuth data", *Geophysics,* **61**, pp.1822-1832.

Foster, I, and Kesselman, C., 1997, "Globus: A Metacomputing Infrastructure Toolkit", *The International Journal of Supercomputing Applications and High Performance Computing*, **11**, issue 2, pp. 115-128.

Groop, W., Doss, N., and Skjellum, A., 1996, "A high-performance, portable implementation of the MPI message passing interface standard", *Parallel Computing*, **22**, no. 6, pp. 789-828.

Karonis, N., Toonen, B., and Foster, I., 2003, "MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface", *JPDC*, **63**, no. 5, pp. 551-563.

Park, K-R, *et al*., 2004, "MPICH GP: A Private-IP-enabled MPI over Grid Environments", *Proceedings of the 2nd International Symposium on Parallel and Distributed Processing Applications*.

Acknowledgement

This work was supported in part by NASA. We take this opportunity to thank Tom Hinke, of NASA and Kevin Walsh of SDSC for their insight and support, and Moritz Fleidner and Mihai Popovici of 3DGeo Development Inc for their contributions.

Air Quality Forecasting on Campus Grid Environment

Yonghong Yan, Barbara M. Chapman, and Babu Sundaram
{yanyh, chapman, babu}@cs.uh.edu

Department of Computer Science
University of Houston

Abstract

Air Quality Forecasting (AQF) is a new discipline that attempts to reliably predict atmospheric pollution. The application has complex workflow and in order to produce timely and reliable forecast results daily, each execution requires multiple computational and storage resources to be simultaneously and collaboratively available. Deploying AQF on grid is one option to satisfy such needs, but requires related grid middleware to support automated application-specific scheduling and execution on grid resources. This paper presents our initial experience of deploying AQF on a campus grid environment and our current efforts of developing a solution of grid-enabling AQF-like applications in Gracce project. Gracce has the goal to provide domain users a grid platform supporting from the management of an application and its dataset, to the automatic execution and viewing of results. In Gracce, application workflow is described using GAMDL, a powerful data-flow language for domain users in describing application logics. The Gracce metascheduler architecture, which includes a workflow-orchestrated metascheduler, an event-driven workflow engine, and an execution runtime system provides the required functionalities of scheduling application workflow in global level and coordinating workflow executions.

1 Introduction

Air Quality Forecasting (AQF) [22] is a new discipline that attempts to reliably predict atmospheric pollution, especially high levels of ozone. The application incorporates multiple dependent computational modules that make intensive use of numerical tools, requires high compute power for the simulation of meteorological and chemical processes, and entails the transfer, storage and analysis of a huge amount of observational and simulation data [6]. We participate in an effort to build such a service, with the goal of providing timely, reliable forecasts of air quality for the Houston-Galveston region and for several other regions in the South Central USA that have encountered problems with air quality in the recent past [3, 4]. On-going work at the University of Houston (UH) aims to create, test and deploy an AQF application as well as to establish a suitable development and deployment environment.

Grid technology [15], and middlewares to enable the creation of such grids, provide a potential strategy for meeting the computational and storage needs of AQF executions. Users with large-scale problems, such as AQF applications, may exploit multiple distributed high performance computing resources in a grid environment to produce high quality results that cannot be achieved from single-domain resources. As the grid technology becomes mature and standardized, deploying application on grid to efficiently use grid powers is becoming more important than technology and standardization themselves. Additional efforts are required to fill the gaps between grid visions and domain expectations. Yet such efforts are still in the stage of trial and related experiences are very application-specific and technology oriented.

Including ours [2, 3], most of current approaches of grid application deployment starts with the packaging or wrapping of legacy application codes with grid services and utilities of grid remote execution and automatic file transfer, and presents them in a grid portal to domain users. Efforts in supporting the automated application-specific scheduling and execution on grid resources, and thus providing users an end-to-end grid environment (not grid technology) are very few. This paper presents our experience of deploying AQF application on a campus grid environment and our current efforts of developing a solution of grid-enabling AQF-like applications in Gracce project [28]. The initial efforts provided a working, but not feature-complete solution to support AQF run on the resources across our campus grid. It is the basis for the next stage development of a general-purpose solution in Gracce.

Gracce has the goal to provide domain users an application grid platform supporting from the management of the application and its dataset, to the automatic execution and viewing of results. In Gracce, application coordination and collaboration, a typical example of which is workflow, is described using GAMDL, a powerful data-flow language for domain users in describing application logics. Gracce metascheduler architecture is designed as a software above the available grid infrastructural middlewares to provide functionalities of grid resource allocation, workflow coordination and runtime control. The architecture includes a workflow-orchestrated metascheduler with planning and reservation features, an event-driven workflow engine able to coordinate the scheduling process and job execution, and a runtime system to control workflow executions.

The organization of this paper is as follows. Section 2 introduces AQF application, our initial efforts in deploying AQF on UH Campus grid [2], and the requirements to support automatic AQF run on grid in production quality. At the end of this section, software and projects related to these requirements are surveyed. Section 3 presents the current two major efforts in Gracce project, GAMDL and Gracce metascheduling architecture. Section 4 summarizes our work and its strengths.

2 Experience of AQF on Campus Grid and New Requirements

Our initial efforts in deploying AQF on grid utilized the basic functionalities provided by Globus toolkits 2.x [12] and provided a working solution to support AQF run on the resources across our campus grid [3]. But it is not feature complete to build an application grid environment, which is our ultimate goal of application deployment on grid. In this section, we introduce AQF application and our current deploying approach, and analyze issues in the approach. Also based on our current experience, three additional features that are required for grid middlewares to fulfill our goal are identified in current stage of the project. Middlewares and efforts related to these features are studied at the end of this section.

2.1 AQF Introduction

AQF is an integrated computational model that is composed of three subsystems: the PSU/NCAR MM5 mesoscale weather forecast model [9], the Sparse Matrix Operator Kernel Emission System code (SMOKE) [24], and EPA's CMAQ chemical transport model [7]. AQF execution is a computational sequence of the three subsystems on heterogeneous resources with increasing resolution and decreasing geographical boundaries. Figure 1 illustrates the workflow of a nested 2-day forecasting operation over a single region of interest by a three-domain computation. The 36km domain computation provides coarse forecast data over continental USA, the 12km provides data across the south central USA, and the 4km forecasts air quality across a smaller geographic region. Each rectangle represents a computational module and each arrow indicates the flow of data between modules. An AQF daily run starts with the download of the data of ETA weather forecast analysis on 15:30PM, and should produce results before 6:00AM the next day to researches and state and local officials [22, 34]. In our experience, a sequential run on a 256-CPU Linux cluster can only finish AQF forecast timely for 12km domain, with about 30G data generated daily. Substantial computational and storage resources are required in order to provide high-quality forecasting in an urban area based upon 4km and 1km domains. Enabling AQF on UH campus grid and utilizing the parallelism of module executions in AQF workflow are two approaches we explored for the timely and accurate forecasting in finer domain regions [2].

2.2 Initial Experience of AQF Deployment on Campus Grid

The UH campus grid currently consists of a heterogeneous cluster of Sun SMPs, a Beowulf cluster and an SGI visualization system, with 9 TB storage, at UH High Performance Computing Center (HPCC) [31], and clusters of Sun SMPs and Beowulf and several Sun workstations in different departments. AQF modules are installed and configured in these resources, and disk and tape spaces are allocated for its daily execution. Sun Grid Engine (SGE) [35] and Platform LSF [32] have been installed to manage resources

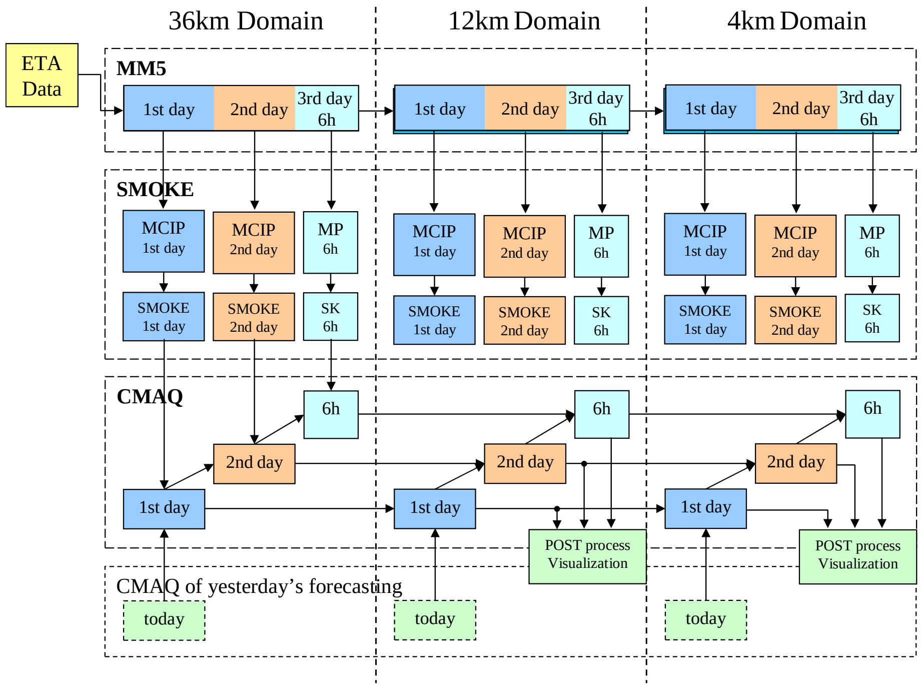


Figure 1: AQF Application Workflow

within the individual administrative domains. Globus toolkits [12] are installed on these resources for grid job execution and file transfer. UH HPCC serves as the CA [14] in our campus grid and is responsible for granting grid accounts. Individual departmental resources are configured to accept only the certificates from this CA. To make it as easy as possible for users to interact with the services provided through the campus grid, EZ-Grid [3], a light-weight web-based portal, have been developed. It uses the Java CoG Kit [11] to provide a convenient interface to all Globus functions, including grid authentication using X.509 certificates and management of GSI proxies [14], job specification, submission and management, file transfers, and grid resource information and load status.

In current campus grid setup, AQF workflow structure is described using an XML file, and a Perl script controls AQF workflow execution and interacts with Globus in EZ-Grid portal. A module in the workflow is described as a task in the XML file that will become a grid computational job (Module, task and job refer to the same entity in different context, the term "module" is from application people, task is a workflow concept and job is often used in grid context). Dependencies between modules are specified as the parent-child relationships of tasks in which parent tasks produce the data that are consumed by child tasks. For each task, details about the executable and resources where it is going to be launched are hard-coded in the task RSL file [36]. The Perl script reads the XML file and controls the overall execution of AQF tasks, including submitting jobs to grid resources, initiating file transfer when the data are available, and resolving task dependencies. So basically, the AQF execution scenario, which is about where, when and how each workflow task is going to be launched and a dependency is handled, are predefined in the description XML file and the control Perl script.

There are several issues in our current solution. Firstly, computational resources are pre-allocated for AQF tasks and are assumed to be available during task execution periods. The allocated resources specified in task RSL file are defined by system administrators, who also reserves the resources in the local scheduler to ensure their availabilities. Obviously, this type of human-scheduling policy is not suitable for the changing grid environments and resource allocation should be automated to provide best decisions according to the resource load status. Secondly, failures in a grid resource will cause the failure of the whole AQF run if without users' intervention. There is no scheduler to allocated resources for a task whose dedicated resource fails. Specifying a secondary resource in the RSL is one solution, yet normally the secondary resource is rather busy such that the task would have to wait for long period in the local queue. Thirdly, the non-standard XML and script approach for workflow description and execution control is error-prone and introduces lots of burden for domain users and system administrators. Domain users are required to be a programmer for XML, Perl and RSL for their applications, which is a daunting task in the process of AQF deployment. Instead of digging down all details of grid setup and resource scheduling in AQF run, users expect a complete application execution environment, from graphical user interface to final view of execution results.

2.3 New Requirements and Studies of Related Grid Middlewares

Understanding the above issues and users' expectations, and also based on our experience in campus grid setup and AQF deployment, we have identified three features that are required from grid middleware to support the automatic AQF execution on grid:

- **Grid Application Modeling and Description** This is to provide domain users a modeling language to describe an application so that users are relieved from the tedious details of workflow, execution control and grid activities. Such language should target for application people, be powerful but easy to use and require only introductory or even no knowledge in grid technology when describing a complex application structure, and should be easily integrated with other grid middlewares, such as workflow and scheduling systems.

- **Grid Metascheduling** AQF daily run requires grid middleware to provide functionalities of automatic resource allocation for AQF workflow tasks across grid. Grid metascheduler operating on the global level is the middleware that may possibly satisfy such needs. But an AQF job typically consists of several dependent tasks and placing these tasks on the appropriate resources across a grid for efficient execution is a much more complex problem than scheduling a single-executable job. Moreover, the scheduling decision must ensure application Quality-of-Service, in our case, which means forecasting results must be generated timely.

- **Workflow Orchestration in Resource Allocations** In workflow execution, the approach of scheduling tasks on resources right after the task dependencies are resolved can not guarantee resources can be discovered and allocated. Resource co-allocation normally requires planning of workflow execution and advanced reservation of resources. In making these decisions, metascheduler should consider the task execution scenario based on AQF workflow to make sure the resource co-allocation are properly coordinated with the application workflow.

There have been lots of efforts addressing issues of scheduling in grid computing area. Globus GRAM [19] and RSL [36] are the early, de-facto standard in providing solutions for secure job execution in metacomputing environments. However, GRAM and Globus itself do not have grid scheduling and brokering functionalities. DUROC [17] is an early effort in Globus 2.x to address the issues of resource co-allocations in RSL-specified multi-request for resources. Globus GARA [13], Maui Silver [33] and architecture defined in [10] introduced advanced reservation [23] into GRAM co-allocations architecture. SNAP [18], which extends Globus GRAM and GARA, proposes a service negotiation protocol into grid scheduling process. Pegasus [8] addresses workflow job scheduling issues as AI planning in constructing and execution workflow from application logic workflow.

Although issues related to grid scheduling have been researched in different projects, efforts to develop a fully functional grid metascheduler are very few, and to our best knowledge, none of them addresses workflow orchestration issue in resource allocations. Community Scheduler Framework (CSF) [26] implements a number of low-level services as a development basis for implementing a fully functional metascheduler. Maui silver [33] scheduler jobs across Maui-managed resources and is not standard based. GRASP [29] aims to provide an OGSI-compliant resource allocation and reservation services following the requirement of Grid Scheduling Architecture proposed by Grid Scheduling Architecture Research Group of GGF [30]. Nimrod/G [20] is an resource management and scheduling system with focus on computational economy in scheduling tasks based on their deadlines and budgets. MARS Metascheduler [1] is an on-demand scheduler which discovers and schedules the required resources for a critical-priority task to start immediately. We also studied efforts in addressing various issues in grid scheduling.

Efforts that address scheduling in workflow community are also very limited. Triana [5] workflow engine schedule tasks across multiple resources either in parallel or in a pipeline. GridFlow [16] executes a grid workflow according to a simulated schedule. Pegasus [8] separates abstract workflow with the concrete workflow and relies on Condor DAGMan [27] to schedule the workflow jobs. For workflow descriptions that directly interest us, several languages are studied based on our AQF needs. Business Process Execution Language (BPEL) [25] is an XML-based workflow definition language that allows businesses to describe enterprise business processes. BPEL is in low-level web service level, which additional extension and wrapping development needed to make it easy of use by grid application owners. XScufl [39] is a specific workflow definition language for Taverna project, but XScufl is too fine grained for describing scientific applications. Abstract Grid Workflow Language (AGWL) [21] "programs" application control flow using constructs of imperative programming style. For data-flow applications, users have to translate the dataflow into control-flow to use AGWL.

3 Gracce: Building An Application Grid Environment

Driven by AQF application, Gracce (Grid Application Coordination, Collaboration and Execution) project [28] was proposed to develop a set of grid middlewares for grid application deployment. The vision of Gracce is to provide domain scientists an application-specific grid environment, supporting from the management of an application and its dataset, to the automatic execution and viewing of results. In Gracce solutions, domain users are only required to provide application descriptions and major resource requirements, and Gracce is responsible for allocating grid resources for tasks, placing tasks on resources for execution and monitoring them, and returning the results back to users. The three required features for automatic AQF execution on grid are addressed by two efforts in Gracce: the Gracce Application Modeling and Description Language, and Gracce metascheduler architecture.

3.1 Gracce Application Modeling and Description Language (GAMDL)

GAMDL is a high level abstraction language for domain scientists to describe their applications in grid environment. GAMDL is a data-flow modeling language, compared with other solutions that describe application control-flow structures. GAMDL models an application in its domain logics and users do not need to extract application control structures to construct a workflow. By using conditioned properties and conditioned pipes, GAMDL allows control-flow to be defined within dataflow. GAMDL also introduces the concept of multiple-value property (mvproperty) for easy description of similar application entities, such as files, modules, and executables. In a GAMDL document, a universal ID (uid) is used to identify and reference an application entity, which may not be defined in the same document. This is very helpful in programming and mapping application entities with persistence services, such as RDBMS, XML and Java Object.

GAMDL is specified using XML-Schema and a grid application is represented as a `gridApp` XML document with four major child elements: `appExecutables, appDataFiles, appModules and appMdDeps`, which specify the required executables, files, modules, and module dependencies respectively in an application. A task in application workflow is modeled as a "module", a term domain users are familiar with. A module, which consists of executables, input/output file set, and its grid job specification, is normally a computation unit that will become a single-executable grid job. Dependency relationships between modules can be specified in either parent-children pattern indexed by parent tasks or child-parents pattern indexed by child tasks. In each relationship, dependencies are specified by pipes, whose pipeIn specifies the piped output of parent tasks, and pipeOut specifies the piped input of child tasks. Each pipe is conditioned by a boolean string that will be evaluated runtimely to decide whether the piped dependency should be handled or not.

The GAMDL description for AQF is attached in appendix. In AQF gridApp XML document, mvproperties are defined by either including from files (`uhaqf.mvproperties` in this example) or defining them directly(`mdName`). The file `uhaqf.mvproperties` defines three mvproperties: `md={mm5,smoke,cmaq}`, `dmsz={36K,12K,4K}`, and `day={d1,d2}`. The `mdName($md,$dmsz,$day)`, defined as `uhaqf-$md-$dmsz-$day` ($ operator on a mvproperty refers to its values), is extended into 18 (which is #`md`*#`dmsz`*#`day`, # operator on a mvproperty returns the number of its values) values. So this one sentence is enough to reference all the AQF computational modules. AppExecutables, appDataFiles and appModules are all defined here as uid reference and they should be specified in other documents. AppMdDeps are specified using child-parent relationships, which is easily understood from the GAMDL itself. For detail about GAMDL, we refer readers to [28].

3.2 Gracce Metascheduling Architecture

We define metascheduler as "a grid middleware that discovers, evaluates and co-allocates resources for grid jobs, and coordinates activities between multiple heterogeneous schedulers that operate at the local or cluster level". There are two aspects covered in this definition, the "scheduling" aspect which addresses resource co-allocation issues for applications requiring resources simultaneously at multiple sites, and the "meta" aspect that concerns about the brokering from global grid requests onto resource local schedulers. To address these two aspects, our metascheduler design separates job execution from the metascheduler, making scheduling process independent from underlying grid middleware for job execution. This allows metascheduler to work with various remote execution utilities. The separation is achieved by the concept of Execution Plan (EP) for a workflow job. The job EP contains scheduling decisions for each task and mechanisms for dependency handling, and a separate runtime system translates the EP into execution-specific scripts for job execution and controls. The defined architecture has three components, Metascheduler, EPExec runtime system, and GridDAG workflow engine. To deploy this architecture in a grid environments, we assumes the installation of Grid Information Services. The complete setup is shown in Figure 2.

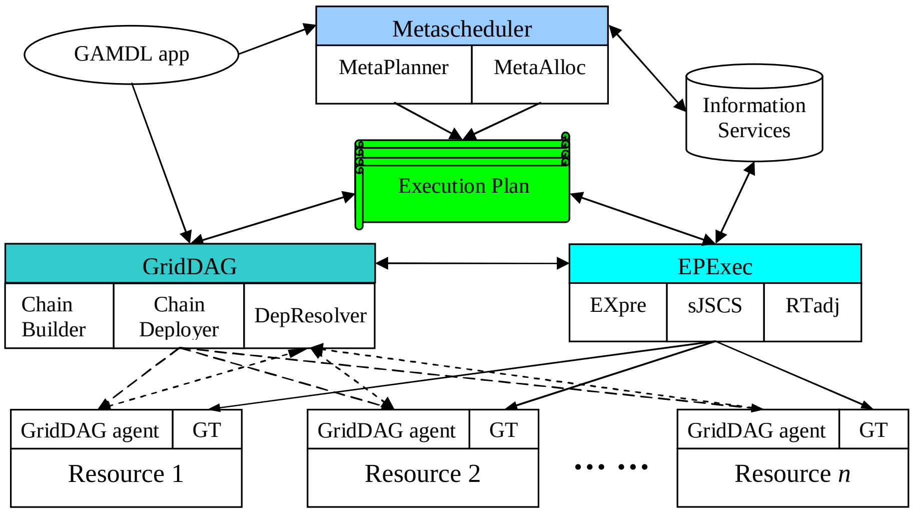


Figure 2: Gracce Metascheduling Architecture

GridDAG is an event-driven workflow system to coordinate the execution of tasks with dependencies. GridDAG events, such as completion of tasks or dependent file availabilities are WS-Notification NotificationMessages produced and consumed by corresponding entities in the architecture. The sequence of these events are referred as event chains and two GridDAG modules, chain builder and deployer setup and install these chains. GridDAG DepResolver keeps track of the events along job execution and invokes certain handlers upon receiving events. GridDAG agents, installed optionally on grid resources generate the outgoing events and consume incoming events locally.

Metascheduler plans job execution and co-allocates resources for workflow tasks. Two modules are designed for these two functionalities, MetaPlanner and MetaAlloc. MetaPlanner predicts the execution scenario for each task; and MetaAlloc searches for suitable resources, negotiates the resource provision and makes reservation with resource providers. The whole metascheduling process is orchestrated by job workflow and the output of this process is a job EP which includes resource allocation decisions and mechanisms for task dependency handling.

EPExec submits task jobs following the EPs, and monitors and manages the execution of these tasks. EPExec also works with GridDAG for handling task dependencies. During execution, EPExec may adjust EP according to the real execution scenario. Being independent from metascheduler, EPExec can be developed to support different methods of job submission and remote execution utilities. EPExec has three components, EXEpre (Execution Preparation), sJSCS (simple Job Submission and Control Service) and RTadj (RunTime Adjuster).

The life-cycle of a grid workflow job in this architecture is described briefly below:

1. Grid users submit to the Metascheduler a workflow job (possibly with preferred deadline) specified using the application GAMDL.

2. Metascheduler plans the execution of the tasks and the dependency handling mechanisms, and allocates resources for tasks executions. As a result, the job's EP that includes the decision details is outputted.

3. The job's EP is forwarded to EPExec for job execution, which submits tasks to their allocated resources and monitors their executions.

4. During execution, GridDAG agents and DepResolver handle task dependencies and decide whether task dependencies are resolved. EPExec also handles failures and makes required adjustments when the executions are not following the plan.

3.2.1 Gracce metascheduler

Gracce metascheduler plans job execution and co-allocates resources for workflow tasks by the two modules, MetaPlanner and MetaAlloc. MetaPlanner predicts and identifies the execution windows for each task and MetaAlloc searches a list of candidate resources, negotiates and makes the agreement with resource owners of resource provision. A task's execution window (EW), represented by a <EWstarttime, EWlength> pair, is a time period in which task execution shall be. EWstarttime is the window starttime and EWlength denotes the length of this window. The metascheduling process is orchestrated by job workflow so that execution orders of dependent tasks are kept and parallelism of execution of tasks without dependencies are employed.

Given a workflow job, the scheduling process starts with the allocation of resources for the first task in the workflow by the MetaAlloc. When resources are allocated, MetaAlloc also identifies the task's EW. Then, metascheduler processes the child tasks of the first task. First, MetaAlloc discovers a list of candidate resources for each child task and calculate the costs of file transfer between the resource for first task and the candidate resources for child tasks. Secondly, MetaPlanner predicts the task EWs on each candidate resource. EWstarttime is calculated by adding three time value together, the EWstarttime and the EWlength of the first task, and the cost for dependency handling; and EWlength is equal to task's wall-clock execution time. Thirdly, the predicted task EWs associated with each candidate resources are processed by MetaAlloc again, which will choose the best resource and finally reserve the resource for each task. Metascheduler then moves on to process other tasks until the last one. For tasks with more than one parent tasks, MetaPlanner considers the one with latest EW in prediction. Brother tasks will normally have overlapped EW, which means that they may be in execution at the same time. In calculating any time value and task EW, certain grace periods or buffer time are applied.

MetaAlloc allocates grid resources for jobs, mainly computational resources for tasks in a sequence of resource discovery, negotiation, and reservation. In resource discovery, MetaAlloc looks up in Grid Information Services the resources that satisfy task resource requirements and are also available during its EW. The process of filtering resource is split into two stages to ultimately identify a list of candidate resources for the given task's specification. In first stage, resources are selected by a simple match-making of each attribute of task's specification with static resource information. The resources on which the task is able to run are picked to be further evaluated according their runtime information. So in second stage, selected resources are checked for their availabilities during task EW and MetaAlloc finally identifies a list of candidate resources. For each of these candidates, MetaAlloc reservation negotiates with the resource local schedulers about resource provision and makes agreement on the availability of resources. For a list of discovered candidate resources, MetaAlloc requests reservation for resources during task's EW and this is considered as a negotiation process. If local schedulers grant this request, MetaAlloc chooses the one that can provide the earliest EW for the task. A reservation ID is returned which will be used to access the reservation. If no reservation could be made for all the candidates, grace periods are added to the EW and MetaAlloc requests reservation again for other wall-clock periods within the EW until a reservation is made. If MetaAlloc cannot reserve resources for the task, metascheduler stops on this task and forwards the partial EP to EPExec to launch the job. During job execution, MetaAlloc attempts the resource allocation steps periodically for this task until decisions are made.

3.2.2 GridDAG Workflow System

GridDAG is our event-driven workflow system able to coordinate the execution of dependent tasks of a workflow job. Events are notification about status change of jobs or file transfers, data availabilities, or other situations defined by users for resource accounting and monitoring purpose. The event producers detects certain situation or change of status, generate the corresponding event messages and distributes them. The event consumers receive an event and then take certain actions or invoke event handlers. The GridDAG event mechanism is developed using WS-Notification standard [38] and the concepts of event, event producers and consumers map to the situation, NotificationProducer, and NotificationConsumer in WS-Notification specification. A Subscriber is an entity that acts as a service requester, sending the subscribe request message to a NotificationProducer.

As shown in Figure 2, four components are designed in GridDAG to support the eventing mechanisms, event chain builder, chain deployer, GridDAG agent, and DepResolver. The chain builder reads job execution plan forwarded from metascheduler and generate the event chains according to the EP. An event chain is a sequence of the events flowing between the participating producer and consumers in the predefined order. The chain builder also decides who are the producer, consumer, and Subscriber; and what events are going to be generated by each producer and to be received by each consumer. The chain deployer sends subscription requests to producers. A Subscription represents the relationship between a consumer, producer, and related event messages. These relationships constitutes the runtime event chains of a GridDAG job. GridDAG agents installed on each grid resources coordinate the runtime event activities in a distributed fashion by playing several roles at the same time. First, as the event producer, detects events occurred on the host resources, and generates and sends out event message. Secondly, as a consumer, receives messages about the availability of dependent files on remote resources or locally, and takes actions accordingly, such as pulling files. Another GridDAG module, DepResolver is configured to received all event notifications and keeps track of the states of tasks' dependencies (a task may have more than one dependencies). When all dependencies of a tasks are resolved, DeResolver takes certain actions, which are typically sending requests to EPExec for job submission or control.

3.2.3 EPExec Runtime system for Job Execution and Management

EPExec executes workflow jobs by carry out its Execution Plan. EPExec has three main functional modules, EXEpre (Execution Preparation), sJSCS (simple Job Submission and Control Service), and RTadj (Runtime Adjuster). EPExec's EXEpre fills in job EP the required information for job submission and workflow control. sJSCS is a simple utility answering requests from EPExec to submit single-executable jobs and control them. EPExec RTadj identifies differences between the job execution and the job EP, and makes certain adjustments on the execution so that it follows the EP.

When a job EP is forwarded to EPExec for execution, EPExec first calls EXEpre to setup execution related details; and then calls sJSCS to submit the job of the first task to its allocated resources, thus begins the execution cycle of the workflow job. Task job is submitted using its resource reservation ID and the task EWstarttime are set in the resource local scheduler. A successful submission returns a global job ID, which EPExec uses for job monitoring and control. During job execution, GridDAG agent and DepResolver work together to handle task dependencies. For data dependencies, file transfer can be in either destination-pull or source-push mode. In destination-pull mode, GridDAG events on source resources sends events about file availabilities to destination GridDAG agent, which then fetches files and sends events to GridDAG DepResolver about file arrivals. For source-push mode, when files are available, source GridDAG agent transfers them to the destination resources and send events to DepResolver indicating that the intermediate files have been transferred. As the overall coordinator of dependency handling, DepResolver keeps track of the status of the dependencies of all tasks and decides whether dependencies of a task are all resolved.

If job execution does not follow the EP, RTadj is responsible to adjust the mismatch to make sure that tasks are executed on the allocated resources during the reserved time frame. RTadj categorizes job execution into four situations depending on how much task executions deviate from the EP. In the first situation, tasks are executing within their EWs and no adjustment is needed. In the second situation, a job completes after its EW, but the difference is within the grace periods of its child tasks so that they all can be started within their EW. In the third situation, the task's late completions go beyond the grace periods of their child tasks and cause their execution not to complete within the resource reservation time frame. Instead of killing those jobs, we configure local scheduler to allows them to finish. Currently, RTadj does not make adjustment for the situation two and three and relies on the allocated buffer time in task EW to automatically repair this. In the fourth situation, the tasks' late completions cause the expiration of reservation of its child tasks. In this case, EPExec submits these jobs without using reservation, yet the jobs may be held in the resource local queues. So, after submitting them, RTadj will request metascheduler to discover other resource for these tasks. If some resources are discovered and allocated, EPExec submits another copies of these tasks to these resources. During execution, EPExec kills the one that it thinks will be completed later than another one. In doing this, RTadj tries its best to make up the lost time in past job execution and minimizes the negative impacts on the execution of later tasks. If cannot make up these delays and it is almost impossible to follow the original plan, RTadj will consider re-scheduling for the rest of tasks. RTadj forwards the sub graph of job GridDAG to Metascheduler to do re-planning and re-allocating. Re-scheduling may cause low resource usage or wasting because of the cancellation of the reservation that has already been made. Metascheduler should avoid such cancellation by scheduling other jobs onto these reservations.

4 Conclusions

AQF is a typical application that requires several computational resources simultaneously and collaboratively available to produce air quality forecasting results in timely fashion. While a grid environment has potential to satisfy such requirement, lots of efforts are still needed to fill the gap between grid community and domain scientists. This paper presents our efforts to provide solutions for domain scientists to enable their applications on grid. Driven by AQF application and based on our past experiences of grid deployment in UH campus grid, the ongoing Gracce project attempts to provide an end-to-end solution for automatic application execution on grid environments. Using middlewares of Gracce, domain scientists are only required to specify their application logic structures and major resource requirements, Gracce is responsible to allocate grid computational resources for application tasks, launch the application and deliver the results back to users.

There are two major efforts in Gracce project in current stage: GAMDL and Gracce metascheduler. GAMDL provides a very intuitive means to model an application for grid computing. GAMDL's dataflow style in describing an application reflects the application original logics, requiring no efforts from users to extract application control-flow. The mvproperty concept makes GAMDL to be very powerful in describing similar application entities and a GAMDL description document is very concise and readable. Gracce metascheduling effort researches grid scheduling issues for workflow jobs, defines the term "Grid metascheduler" and proposes a workflow-orchestrated metascheduling architecture. The architecture integrates solutions to scheduling related issues in grid area, such as resource co-allocation, service negotiation and resource reservation, and workflow execution planning.

References

[1] A. Bose, B. Wickman, and C. Wood *MARS: A Metascheduler for Distributed Resources in Campus Grids,* Proceedings of Fifth IEEE/ACM International Workshop on Grid Computing, 2004

[2] B.M. Chapman, P. Raghunath, B. Sundaram, Y. Yan, *Air Quality Prediction in a Production Quality Grid Environment,* Engineering the Grid: status and perspective, edited by J. Dongarra, H. Zima, A. Hoisie, L. Yang and B.D. Martino, Spring 2005

[3] B.M. Chapman, H. Donepudi, J. He, Y. Li, P. Raghunath, B. Sundaram and Y.Yan, *Grid Environment with Web-Based Portal Access for Air Quality Modeling,* Parallel and Distributed Scientific and Engineering Computing, Practice and Experience, 2003

[4] B.M. Chapman, Y. Li and B. Sundaram, and J. He, *Computational Environment for Air Quality Modeling in Texas,* Use of High Performance Computing in Meteorology, World Scientific Publishing Co, 2003

[5] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, *Programming Scientific and Distributed Workflow with Triana Services,* Special Issue of Concurrency and Computation: Practice and Experience? 2005

[6] D. W. Byun, J. Pleim, R. Tang, and A. Bourgeois, *Meteorology-Chemistry Interface Processor (MCIP) for Models-3 Community Multiscale Air Quality (CMAQ) Modeling System,* Washington, DC, U.S. Environmental Protection Agency, Office of Research and Development, 1999.

[7] D. W. Byun, K. Schere, *EPA's Third Generation Air Quality Modeling System: Description of the Models-3 Community Multiscale Air Quality (CMAQ) Model,* Journal of Mech. Review, 2004

[8] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, and M. Livny, *Pegasus: Mapping Scientific Workflows onto the Grid,* Across Grids Conference 2004, Nicosia, Cyprus

[9] G. Grell, J. Dudhia, and D. Stauffer, *A Description of the Fifth-Generation Penn State/NCAR Mesoscale Model (MM5) NCAR/TN-398+STR,* NCAR Tech Notes http://www.mmm.ucar.edu/mm5/

[10] G. Mateescu, *Quality of Service on the Grid Via Metascheduling with Resource Co-Scheduling and Co-Reservation,* International Journal of High Performance Computing Applications, Vol. 17, No. 3, 209-218 (2003)

[11] G. von Laszewski, I. Foster, J. Gawor, and P. Lane, *A Java Commodity Grid Kit,* Concurrency and Computation: Practice and Experience, vol. 13, no. 8-9, pp. 643-662, 2001, http:/www.cogkits.org/

[12] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit,* International Journal of Supercomputer Applications, Summer 1997.

[13] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, and A. Roy, *A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation.* Intl Workshop on Quality of Service, 1999

[14] I. Foster, C. Kesselman, G. Tsudik, S. Tuecke, *A Security Architecture for Computational Grids,* ACM Conference on Computers and Security, 1998, 83-91.

[15] I. Foster, C. Kesselman, S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations,* International Journal of High Performance Computing Applications, 15 (3). 200-222. 2001.

[16] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd, *GridFlow: Workflow Management for Grid Computing,* In Proceedings of 3rd International Symposium on Cluster Computing and the Grid, at Tokyo, Japan, May 12-15, 2003, p. 198.

[17] K. Czajkowski, I. Foster, and C. Kesselman, *Resource Co-Allocation in Computational Grids,* Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC-8), pp. 219-228, 1999.

[18] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, *SNAP: A Protocol for negotiating service level agreements and coordinating resource management in distributed systems,* Lecture Notes in Computer Science, 2537:153-183, 2002.

[19] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, *A Resource Management Architecture for Metacomputing Systems,* Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, pg. 62-82, 1998.

[20] R. Buyya, D. Abramson, and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid,* The 4th International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia 2000), May 2000

[21] T. Fahringer, J. Qin and S. Hainzer, *Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language,* Proceedings of Cluster Computing and Grid 2005 (CCGrid 2005)

[22] W. F. Dabberdt, M. A. Carroll, D. Baumgardner, G. Carmichael, and R. Cohen *Meteorological research needs for improved air quality forecasting,* Report of the 11th Prospectus Development Team of the U.S. Weather Research Program, 2004.

[23] W. Smith, I. Foster, and V. Taylor, *Scheduling with Advanced Reservations.* Proceedings of the IPDPS Conference, May 2000.

[24] Z. Adelman and M. Houyoux, *Processing the National Emissions Inventory 96 (NEI96) version 3.11 with SMOKE,* The Emission Inventory Conference: One Atmosphere, One Inventory, Many Challenges, 1-3 May, Denver, CO, U.S. Environmental Protection Agency, 2001.

[25] BPEL4WS: Business Process Execution Language for Web Services Version 1.0, http://www.106.ibm.com/developerworks/webservices/library/wsbpel

[26] Community Scheduler Framework, http://www.platform.com/products/Globus/

[27] DAGMan (Directed Acyclic Graph Manager), http://www.cs.wisc.edu/condor/dagman.

[28] Gracce: Grid Application Coordination, Collaboration and Execution, http://www.cs.uh.edu/˜yanyh/gracce

[29] Grid Resource Allocation Services Package (GRASP), http://www.moredream.org/grasp.htm

[30] Grid Scheduling Architecture Research Group, https://forge.gridforum.org/projects/gsa-rg

[31] High Performance Computing Center, University of Houston, http://www.hpcc.uh.edu

[32] Load Sharing Facility, Resource Management and Job Scheduling System, http://www.platform.com/products/HPC/

[33] Maui Moab Grid Scheduler (Silver), http://www.clusterresources.com/products/mgs/

[34] NCEP ETA analysis and forecast http://www.emc.ncep.noaa.gov, http://www.emc.ncep.noaa.gov/data

[35] Sun Grid Engine, Sun Microsystems, http://www.sun.com/software/gridware

[36] The Globus Resource Specification Language RSL v1.0, http://www-fp.globus.org/gram/rsl_spec1.html

[37] UH IMAQs - Institute for Multidimensional Air Quality Studies, http://www.imaqs.uh.edu

[38] Web Services Notification, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn

[39] XScufl Language Reference, http://taverna.sourceforge.net/docs/xscuflspecification.html

Appendix: GAMDL Description for AQF Application

```
<gridApp uid="uhaqf05" xmlns="http://.../gamdl"
     xsi:schemaLocation="... gamdl.xsd">
 <name>UH-AQF-2005</name>

 <mvproperty file="uhaqf.mvproperties"/>
 <mvproperty name="mdName($md,$dmsz,$day)">
  <value>uhaqf-$md-$dmsz-$day</value>
 </mvproperty>
 <include href="aqfexe.xml"/>
 <include href="aqffiles.xml"/>
 <include href="aqfmd.xml"/>

 <appExecutables>
  <executable uidRef="uhaqf-$md"/>
 </appExecutables>

 <appDataFiles>
  <file uidRef="$mdName($md,$dmsz,$day)-in1"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-in2"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-in3"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in2"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in3"/>
  <file uidRef="$mdName(cmaq,$dmsz,$day)-in4"/>
  <file uidRef="$mdName($md,$dmsz,$day)-out1"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-out2"/>
  <file uidRef="$mdName(mm5,$dmsz,$day)-out3"/>
  <file uidRef="$mdName(smoke,$dmsz,$day)-out1"/>
  <file uidRef="$mdName(smoke,$dmsz,$day)-out2"/>
  <file uidRef="uhaqf-postpv-$day-in1"/>
  <file uidRef="uhaqf-postpv-$day-out1"/>
 </appDataFiles>

 <appModules>
  <module uidRef="$mdName($md,$dmsz,$day)"/>
  <module uidRef="uhaqf-postpv-$day"/>
 </appModules>

 <appMdDeps>
  <CPsRship uid="smoke-mm5-$dmsz-$day-CPs">
   <childMd uidRef="$mdName(smoke,$dmsz,$day)"/>
   <parentMd uidRef="$mdName(mm5,$dmsz,$day)">
    <viaPipe>
     <pipeInFile uidRef="$mdName(mm5,$dmsz,$day)-out1"/>
     <pipeOutFile uidRef="$mdName(smoke,$dmsz,$day)-in1"/>
    </viaPipe>
   </parentMd>
  </CPsRship>
  <CPsRship uid="cmaq-smoke-$dmsz-$day-CPs">
   <childMd uidRef="$mdName(cmaq,$dmsz,$day)"/>
   <parentMd uidRef="$mdName(smoke,$dmsz,$day)">
    <viaPipe>
     <pipeInFile uidRef="$mdName(smoke,$dmsz,$day)-out1"/>
     <pipeOutFile uidRef="$mdName(cmaq,$dmsz,$day)-in1"/>
    </viaPipe>
    <viaPipe>
     <pipeInFile uidRef="$mdName(smoke,$dmsz,$day)-out2"/>
     <pipeOutFile uidRef="$mdName(cmaq,$dmsz,$day)-in2"/>
    </viaPipe>
   </parentMd>
  </CPsRship>

  <CPsRship uid="mm5-12k-36k-$day-CPs">
   <childMd uidRef="uhaqf-mm5-12k-$day"/>
   <parentMd uidRef="uhaqf-mm5-36k-$day">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-mm5-36k-$day-out3"/>
     <pipeOutFile uidRef="uhaqf-mm5-12k-$day-in3"/>
    </viaPipe>
   </parentMd>
  </CPsRship>
  <CPsRship uid="mm5-4k-12k-$day-CPs">
   <childMd uidRef="uhaqf-mm5-4k-$day"/>
   <parentMd uidRef="uhaqf-mm5-12k-$day">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-mm5-12k-$day-out3"/>
     <pipeOutFile uidRef="uhaqf-mm5-4k-$day-in3"/>
    </viaPipe>
   </parentMd>
  </CPsRship>
  <CPsRship uid="cmaq-12k-36k-$day-CPs">
   <childMd uidRef="uhaqf-cmaq-12k-$day"/>
   <parentMd uidRef="uhaqf-cmaq-36k-$day">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-cmaq-36k-$day-out2"/>
     <pipeOutFile uidRef="uhaqf-cmaq-12k-$day-in4"/>
    </viaPipe>
   </parentMd>
  </CPsRship>
  <CPsRship uid="cmaq-4k-12k-$day-CPs">
   <childMd uidRef="uhaqf-cmaq-4k-$day"/>
   <parentMd uidRef="uhaqf-cmaq-12k-$day">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-cmaq-12k-$day-out2"/>
     <pipeOutFile uidRef="uhaqf-cmaq-4k-$day-in4"/>
    </viaPipe>
   </parentMd>
  </CPsRship>

  <CPsRship uid="mm5-2d-1d-$dmsz-CPs">
   <childMd uidRef="uhaqf-mm5-$dmsz-2d"/>
   <parentMd uidRef="uhaqf-mm5-$dmsz-1d">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-mm5-$dmsz-1d-out2"/>
     <pipeOutFile uidRef="uhaqf-mm5-$dmsz-2d-in2"/>
    </viaPipe>
   </parentMd>
  </CPsRship>
  <CPsRship uid="cmaq-2d-1d-$dmsz-CPs">
   <childMd uidRef="uhaqf-cmaq-$dmsz-2d"/>
   <parentMd uidRef="uhaqf-cmaq-$dmsz-1d">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-cmaq-$dmsz-1d-out1"/>
     <pipeOutFile uidRef="uhaqf-cmaq-$dmsz-2d-in3"/>
    </viaPipe>
   </parentMd>
  </CPsRship>

  <CPsRship uid="postpv-cmaq4k-$day-CPs">
   <childMd uidRef="uhaqf-postpv-$day"/>
   <parentMd uidRef="uhaqf-cmaq-4k-$day">
    <viaPipe>
     <pipeInFile uidRef="uhaqf-cmaq-4k-$day-out2"/>
     <pipeOutFile uidRef="uhaqf-postpv-$day-in1"/>
```

```
    </viaPipe>
   </parentMd>
  </CPsRship>
 </appMdDeps>

 <startMdUid>uh-aqf-mm5-36k-1d</startMdUid>
</gridApp>
```

Experiences from Simulating the Global Carbon Cycle in a Grid Computing Environment

Jason Cope*, Craig Hartsough†, Sean McCreary*, Peter Thornton†, Henry M. Tufo*†, Nathan Wilhelmi† and Matthew Woitaszek*

* University of Colorado, Boulder
† National Center for Atmospheric Research
jason.cope@colorado.edu

Abstract. We discuss our software development experiences with Grid-BGC, a grid-enabled terrestrial carbon cycle modeling environment. Grid-BGC leverages grid computing technologies to create a secure, reliable and easy to use distributed computational environment for climate modeling. The goal is to develop a system which insulates the scientists from tedious configuration details thereby increasing scientific productivity. This project is part of a collaborative effort between the University of Colorado and the National Center for Atmospheric Research to create a general grid-enabled computational framework for climate modeling. Over the course of this project we gained valuable experience deploying grid technology and learned how to create a production quality grid system. We provide an overview of our current system, describe our most salient experiences, and present a proposed production architecture for Grid-BGC.

Introduction

Grid-BGC is a grid-enabled global carbon cycle modeling system and computational framework. Using grid computing technologies, such as the Globus Toolkit [5], researchers and software engineers at the National Center for Atmospheric Research (NCAR) and the University of Colorado at Boulder (CU) implemented a prototype of a grid-enabled system which models the global carbon cycle using computational and storage resources distributed between both institutions. The ultimate goal of this system is to give scientists access to the climate models and data necessary to model the carbon cycle and minimize the complexity of running the models in a distributed computational environment.

In creating this system we gained many experiences on how to develop a stable and usable grid computing system. While NCAR actively participates in archiving climate related data on the grid (e.g., the Earth System Grid) the Grid-BGC project is the first grid-enabled modeling environment developed and deployed by NCAR and provided as a grid service to the climate modeling community. Throughout this project, we gained valuable experience constructing a distributed climate modeling environment using grid technologies. These experiences ranged from mechanical practices, such as using the Globus toolkit, to engineering practices, such as defining a sound security model.

This paper discusses the relevant experiences and lessons learned during the development of the system prototype. We provide a detailed description of the prototype along with a description of how our experiences and lessons learned influenced the final production system's design. We start with an overview or related work, then provide a brief introduction to carbon cycle modeling, followed by an overview of the prototype architecture and implementation of the Grid-BGC system. The following section discusses our experiences and lessons learned during development and operation of the prototype. Finally, we present our proposed production architecture, future work, and conclusions.

Related Work

As the grid computing paradigm matures, more organizations are utilizing the grid computing software infrastructure to help support their scientific and computational needs. In [2], we presented an overview of similar projects (e.g., GEMCLA [6], DIRAC [14], and NorduGrid [4]). We concluded that the Grid-BGC project differed from these projects because our system provides users with a simple interface to request execution of a limited set of climate models on their behalf and guarantees that all aspects of the requested computation are accomplished with complete transparency to the scientist. In contrast, these other solutions are intended for general workflow processing and provide a generic language and parser to define and manipulate workflows, at the cost of introducing substantial complexity to the system design. Two recent external projects similar to Grid-BGC are GridChem and CRAFT.

GridChem [10] is a collaborative project between the NCSA, OSC, and TACC to interface chemists' desktop computers with a grid-enabled environment. The researchers found that the grid computing model and software is difficult for users to adjust to and addressed this in their systems design. GridChem users interact with a Java client, locally installed on users' desktops. The client communicates with a middleware server to authorize and authenticate users as well as manage their workflow tasks. The middleware server implements a customized data management scheme and utilizes Condor [8] for task scheduling. Grid-BGC differs from GridChem in several ways. Grid-BGC utilizes a centralized web-client interface and does not require the installation of a client on the users' desktops. Grid-BGC also focuses on the development of a reliable and automated computational fabric.

Project CRAFT [3] is another collaborative project that utilizes grid computing middleware to link remote resources. The ultimate goal of CRAFT is to link data collected from remote sensing instruments into event-driven meteorological models in real-time. CRAFT utilizes grid middleware to create a virtual machine room for computing the models as separate computational centers. Like Grid-BGC, CRAFT utilizes grid computing resources to offload load the execution of meteorological models.

Terrestrial Ecosystem Modeling

Modeling the carbon cycle is accomplished through a multistage workflow composed of two climate models; Daymet and Biome-BGC (see Figure 1). This workflow transforms observed meteorological and ecosystem data, gathered from a finite number of observation stations over the past 50 years, into a high resolution grid of data. With the gridded-weather data, the workflow simulates the carbon cycle on each point in the gridded data set and produces another grid of the simulated carbon cycle data. The generated carbon cycle data can be analyzed through text analysis or visualization tools.

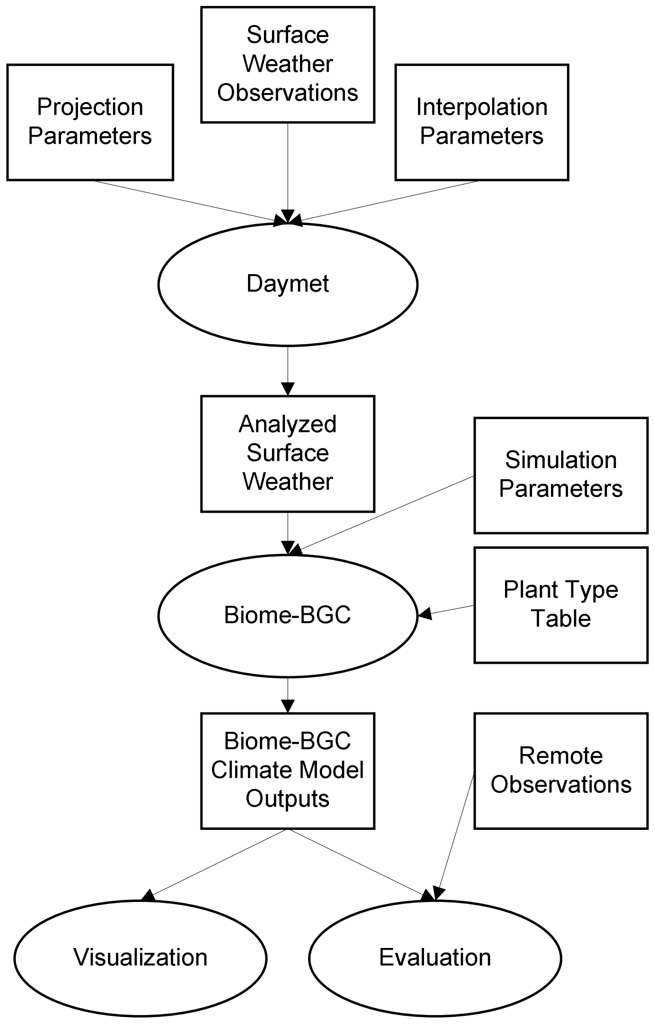


Figure 1, Grid-BGC Workflow

The first transformation performed by the workflow is accomplished through the use of the Daymet climate model [12]. Daymet interpolates historical weather data to produce a high resolution spatial grid of ground-based weather observations. These grids are subdivided into sections referred to as tiles. After the creation of the tiles, the carbon cycle for each tile is modeled using Biome-BGC [13]. This model ingests the tile of climate data along with known soil and plant data and other simulation parameters to simulate the carbon cycle for this tile over a period of time. Post processing of the data is performed to glean data relevant for a particular scientists needs from the output from Biome-BGC.

The point and tile based nature of Daymet and Biome-BGC accommodates simulations of small areas well, but quickly becomes overwhelming for scientists to manage on larger scales. These simulations are embarrassingly parallel because simulations of different tiles can be run in parallel with others. Large area simulations are performed by executing many of the point-based simulations as a collection. In order to achieve a high resolution simulation of a large area, scientists are required to manage many simulations that compose a collection. Management of these simulations requires tedious attention to detail, including periodically monitoring running simulations, transferring data, correctly scripting configuration files for each model, and detecting failed simulations and handling the failures as appropriate. The management of these tasks is further complicated by our computational environment, as resources available to this project are located at two distinct locations: CU houses the allocated computational cluster and NCAR manages the storage systems and web portal user interface.

Prototype Architecture

The goal of our prototype architecture was to address the issue of global carbon cycle modeling complexity in our computational environment. The implementation of our prototype architecture is a federation of several independently functioning components (see Figure 2). These

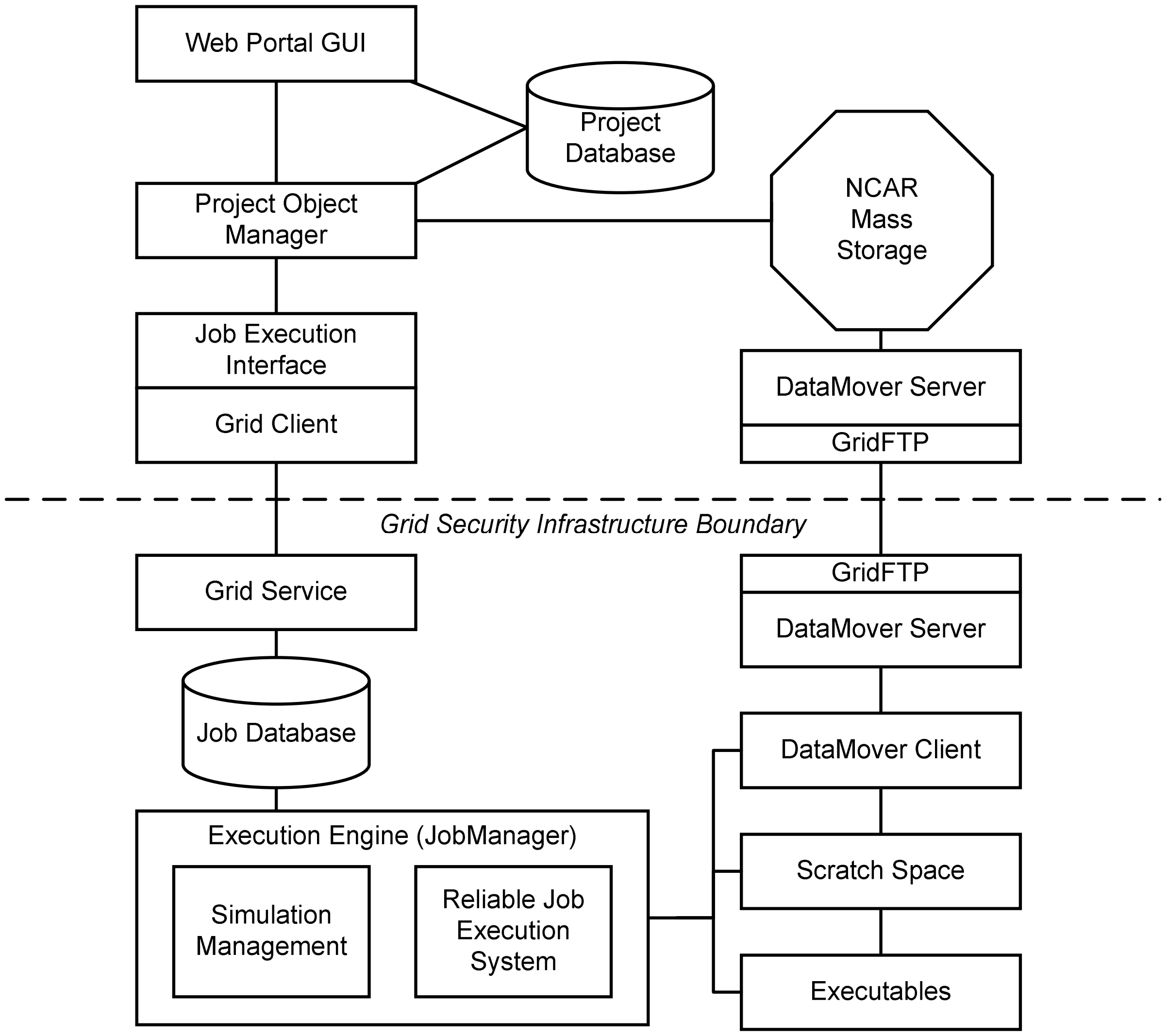


Figure 2, Grid-BGC prototype architecture

components include a web portal, a job management daemon, and a reliable data transfer utility.

The web portal provides a management interface to the Grid-BGC environment for both users and system administrators. The web-portal implementation requires no Grid-BGC specific software packages installed on the client machines since the web portal is maintained at a centralized location. The web portal allows users to specify simulation parameters, manage and monitor their simulations that are in progress, analyze results, and share results with other system users. A client embedded in the web portal communicates with the remote execution environment by invoking methods of a grid service residing on a computational resource. These methods include functionality to stop, start, and monitor simulations in the remote execution environment.

The job management daemon, known as the Grid-BGC JobManager, is used to monitor and coordinate the tasks executing in the computational environment. JobManager runs on the computational resources, such as a cluster. As the grid service receives communications from the web portal, the service parses and stores the requests into a persistent database. JobManager polls the database for new tasks to perform on the environment and the Grid-BGC tasks. The state of Grid-BGC simulations is periodically stored in the same database, so the grid service can query the database when task monitoring information is requested from the web portal. The decoupling of the grid service and JobManager and the use of a database to store system state allows system administrators to arbitrarily restart either component without loss of state, prior to the occurrence of a system failure.

The final component of the system is a data management utility, DataMover [11]. The utility was developed for the Earth System Grid and provides reliable file transfers, data caching, authentication, and interoperability with different storage architectures. DataMover uses GridFTP [1] as the underlying file transfer utility and implements several storage system access protocols, including a protocol to access NCAR's Mass Storage System (MSS). The Grid-BGC prototype uses this tool exclusively to transfer data between CU and NCAR. As the simulations are prepared, DataMover transfers the required input data sets to the computational facility at CU. Once the computations have completed, DataMover transfers the data back to NCAR for storage on the MSS or cached on the grid-enabled host at NCAR.

Experiences and Lessons Learned Implementing the Grid-BGC Prototype

Our prototype architecture successfully implements an end-to-end computational environment for Grid-BGC. The prototype currently manages a small portion of the workflow: the execution of the Biome-BGC model and the associated setup and finalization tasks. During the development of the prototype, we encountered several design problems that helped strengthen our future architecture of the system. Our experiences implementing the prototype include the need to implement a robust security scheme to accommodate the participating institutions security requirements, the need to create a reliable execution environment, and good practices for developing a grid-enabled computational environment.

Security Considerations

As is the case with many grid computing systems, a secure system is critical. While the grid computing middleware provides authentication and encryption mechanisms through the GSI security model, more security measures were needed. For example, NCAR not only requires that access to the MSS be authenticated, it also requires a great deal of accountability for each users actions. A significant challenge in the prototype implementation was to accommodate the rigorous security requirements imposed by our organizations.

Grid computing system security is continually evolving and unfortunately must be reevaluated frequently. This includes addressing more advanced security needs and requirements posed by our organizations, such as one time password authentication, and the policies impacts on our system design. We have found that creating a flexible system design that can accommodate changes in security policy is essential in grid computing development. The flexibility allows security infrastructure to change with minimal impacts to other components of the system.

Reliability and Fault Tolerant Considerations

An essential quality of our computational environment is reliable execution of computational tasks. Fault tolerance in grid computing is being addressed in several areas, including workflows, data management, and task management. We found that integrating fault tolerant capabilities into our system software and grid services, in addition to the previously mentioned areas, strengthened our system design. Our grid service, daemons, and user submitted tasks store little state in volatile memory and log all critical state to a persistent database. With this design, these components can recover from faults by recovering the most recent state from the database and proceeding from this point.

Grid Computing System Development

Grid computing and the development of a grid-enabled environment was a surprisingly more difficult paradigm to become familiar with and acclimated to than we originally expected. We found that pleasant persistence in the face of frustration is an essential quality for the successful development of an environment similar to ours. The Globus toolkit provides many tools to grid developers, but usually at the cost of a complex programming model. From our experiences, a development environment capable of automating code generation for grid components substantially increases development productivity.

We also found that a developer's mileage with the grid middleware will vary. The middleware provides a set of tools that help cope with most grid computing systems, but the development of additional tools to support our application's needs was necessary. Instead of leaving the model configuration for the user to define, we found that development of a tool to automate the creation of configuration files reduced the possibility of human error and interaction with the system. We also found that it was most useful to utilize as much of the established grid middleware as possible. Use of these tools will allow our environment to be re-deployed and interoperate with other environments more easily.

Shortcomings of the Prototype Architecture

From our experiences developing and deploying the prototype, we identified several weaknesses in its design. The prototype architecture deploys a monolithic grid service and daemon that jointly perform all system tasks. A better design would break apart the distinct functions provided by these two components into several modular components. This approach would separate functionality and make the components more extensible for other uses. We believe that staging temporary data to the NCAR MSS is a misuse of the storage system. Only those files that need to be archived should be transferred to the MSS. The prototype is not completely Globus compliant. It uses a non-standard data management utility, DataMover, and Grid-BGC JobManager for execution management. A better approach would utilize the standard components and minimize the software deployment to the climate models and the system services. Porting the prototype architecture would be easier if the third-party utilities were replaced by those that are packaged with the Globus Toolkit.

Production Architecture

Our production architecture addresses the experiences and lessons learned from the implementation of the prototype (see Figure 3). The fundamental goals of our prototype still are the same for the proposed production architecture: the system should be easy for scientists to use and efficiently execute global carbon cycle models. The significant advances we propose for the production architecture is the integration of more Globus Toolkit compliant services into our system, re-structuring data management policies, breaking apart the monolithic service structure to be more service oriented and modular, and re-developing the system from the most recent release of the Globus Toolkit.

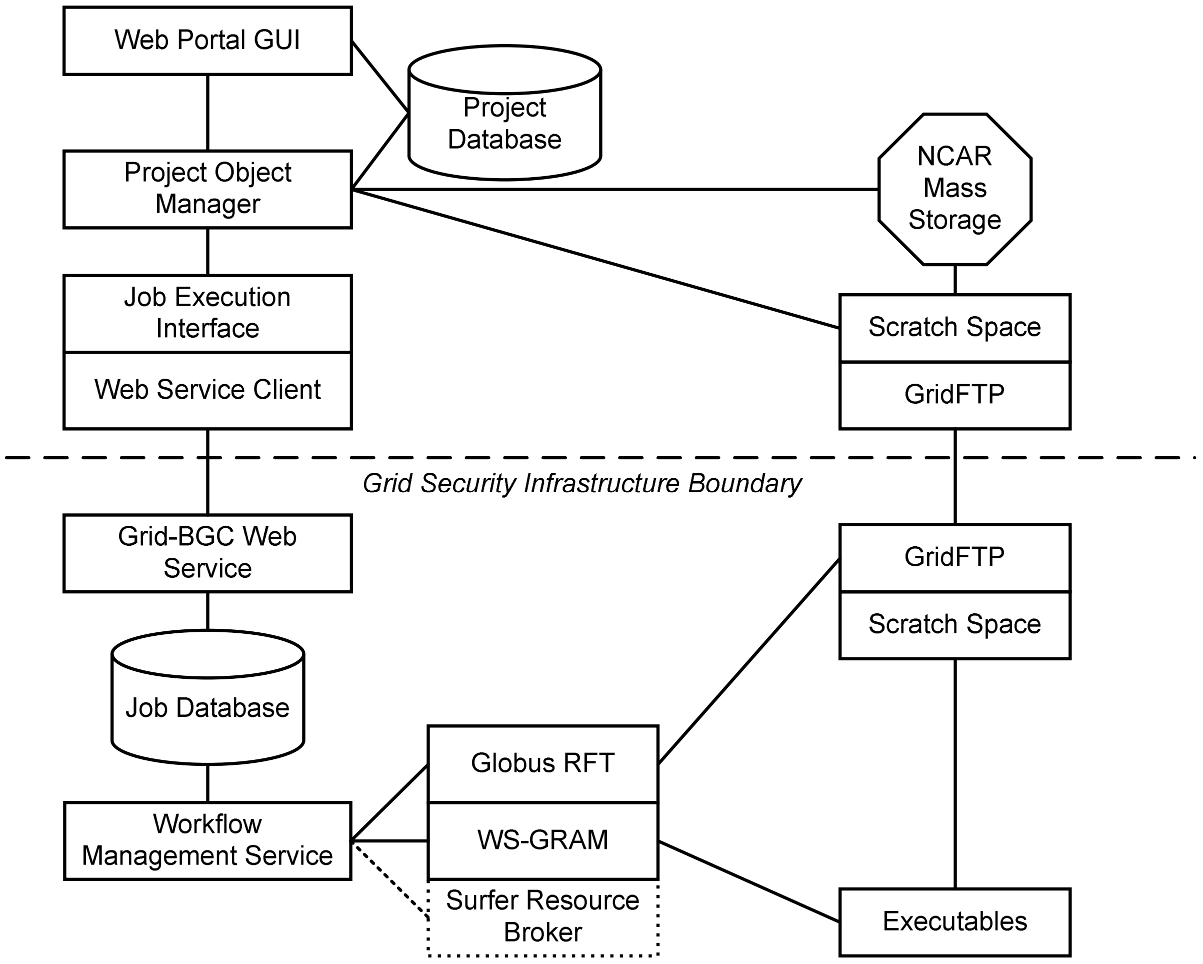


Figure 3, Proposed Grid-BGC production architecture

Our production system will take advantage of several standard services provided by the Globus Toolkit. Prior critiques of the systems design indicated that the use of WS-GRAM would help standardize the execution functionality provided by our system. Our production system will deploy a model execution web service using GRAM as the execution management engine. The service will provide the necessary mechanisms to setup and finalize the execution of the model specified in the web service configuration. WS-GRAM will be supplemented by the fault tolerant, reliable job execution, and management functionality developed during the prototype implementation. We believe that the integration of GRAM into our execution engine will make deployment of our system onto other grids much easier. Additionally, the use of the Globus data management will make our system more interoperable. We are also considering the integration of the Surfer [7] resource broker to help our user community and system allocate available resources as our environment expands from a couple of computational domains to several.

Our objective for restructuring the data management policies and functionality developed for the prototype are to create a more efficient data grid. First, we have decreased our reliance on the NCAR MSS as a staging platform. Instead of staging files through archival storage, we propose staging files using SAN-based scratch space at NCAR and manually archiving files to the MSS as needed. Additionally, we are considering removing DataMover from the system architecture. We believe we can replicate the essential qualities of it by using GridFTP, the Replica Location service, the Reliable File Transfer service, and the Data Storage Interface. Removing DataMover would eliminate the use of a third-party tool from our system design and enable us to use the more recent Globus compliant tools.

The production design also aims to make our system more modular and service oriented. Instead of a single web service to handle all tasks with the system, we propose to modularize the current monolithic grid service. We plan to break out services, such as security, file transfer, data management, job execution, and system management, from the single daemon implemented in the prototype. We believe that breaking out the services has many benefits including reducing the number of bottlenecks, making the system more manageable and scalable, and making the framework more accommodating for use by other grid-based projects in the climate modeling community.

Finally, the use of Globus Toolkit 4 (GT4) in our production prototype should help improve our overall system design for future use. GT4 is web service compliant, so re-development of our services to the standard web service interface will increase interoperability of Grid-BGC with other web service technologies. MyProxy [9] is now a standard component of GT4 and its addition will allow our security requirements to assimilate to other grid computing environments more easily. Significant improvements were made to the data management functionality of GT4, including striped server support for GridFTP, reliable file transfer through RFT, and modular support for non-GridFTP compliant interfaces. These improvements to GridFTP will enable us to better utilize our high-performance storage system through the use of striped servers and mitigate our reliance on DataMover through reliable file transfer support and the development of a modular interface to the NCAR Mass Storage System.

Future Work and Conclusions

Our future work includes developing the production system, deploying the system, and using the system to model the carbon cycle. At this time, we plan on completing the development and deployment of our system by the end of August 2005. Many of the components of the prototype are still usable, including the web portal and many of the reliable execution components. Most of the production development will consist of porting code to GT4 and integrating new services into the system.

We found that the development of the Grid-BGC prototype has been productive, providing experience with grid computing development, security, and reliability. The prototype also highlighted weaknesses in our system design and helped us address these weaknesses in our proposed production architecture. We anticipate our user community will utilize the system to perform scientific studies by the end of October 2005.

Acknowledgements

University of Colorado computer time was provided by equipment purchased under DOE SciDAC Grant #DE-FG02-04ER63870, NSF ARI Grant #CDA-9601817, NSF sponsorship of the National Center for Atmospheric Research, and a grant from the IBM Shared University Research (SUR) program. NASA has provided funding for the Grid-BGC project through the Advanced Information Systems Technology Office (NASA AIST Grant #NAG2-1646) and the Terrestrial Ecology Program.

References

1. Allcock, B., Bester J., Bresnahan, J., Chervenak, A. L., Foster, I., Kesselman, C., Meder, S., Nefedova, V., Quesnal, D., Tuecke, S. Data Management and Transfer in High Performance Computational Grid Environments. Parallel Computing Journal, Vol. 28 (5), May 2002.
2. Cope, J., Hartsough, C., Thornton, P., Tufo, H. M., Wilhelmi, N., Woitaszek, M. Grid-BGC: A Grid-Enabled Terrestrial Carbon Cycle Modeling System, Euro-Par 2005, August 2005.
3. Droegemeier, K.K., K. Kelleher, T. Crum, J.J. Levit, S.A. Del Greco, L. Miller, C. Sinclair, M. Benner, D.W. Fulker, and H. Edmon, 2002: Project CRAFT: A test bed for demonstrating the real time acquisition and archival of WSR-88D Level II data. Preprints, 18th Int. Conf. on Interactive Information Processing Systems (IIPS) for Meteorology, Oceanography, and Hydrology., 13-17 January, Amer. Meteor. Soc., Orlando, Florida, 136-139.
4. Eerola, P., Kónya, B., Smirnova, O., Ekelöf, T., Ellert, M., Hansen, J. R., Nielsen, J. L., Wäänänen, A., Konstantinov, A., Ould-Saada, F. The NorduGrid Architecture and Tools. Proceedings of Computing in High-Energy and Nuclear Physics (CHEP 03), La Jolla, California, March 2003.
5. Globus. The Globus Project, 2004, http://www.globus.org/A
6. Kacsuk, P., Goyeneche, A., Delaitre, T., Kiss, T., Farkas, Z., and Boczko, T. High-level Grid Application Environment to Use Legacy Codes as OGSA Grid Services. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004), Pittsburgh, USA, 8 November 2004.
7. Kolano, P. Z. Surfer: An Extensible Pull-Based Framework for Resource Selection and Ranking. International Symposium on Cluster Computing and Grid 2004, April 2004.
8. Litzkow, M., Livny, M., and Mutka, M. Condor - A Hunter of Idle Workstations, Proceedings of the 8th International Conference of Distributed Computing Systems, pgs 104-111, June, 1988.

9. Novotny, J., Tuecke, S., Welch, V. An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
10. Milfeld, K., Guiang, C., Pamidighantam, S., and Giuliani, J. Cluster Computing through an Application-oriented Computational Chemistry Grid. LCI: Linux Revolution 2005, May, 2005.
11. Sim, A. J. Gu, A. Shoshani, V. Natarajan. DataMover: Robust Terabyte-Scale Multi-File Replication over Wide-Area Networks. Proceedings of the 16th International Conference on Scientific and Statistical Database Management, 403, 21 June 2004.
12. Thornton, P.E., S.W. Running, and M.A. White. Generating surfaces of daily meteorological variables over large regions of complex terrain. Journal of Hydrology, 190: 214-251, 1997.
13. Thornton, P.E., S.W. Running. An improved algorithm for estimating incident daily solar radiation from measurements of temperature, humidity, and precipitation. Agricultural and Forest Meteorology, 93: 211-228, 1999.
14. Tsaregorodtsev, A., Garonne, V., and Stokes-Rees, I. DIRAC: A Scalable Lightweight Architecture for High Throughput Computing. Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing (GRID 2004), Pittsburgh, USA, 8 November 2004.

Grid Application Programming Environments: Comparing ProActive, Ibis, and GAT

Extended Abstract[*]

Thilo Kielmann, Andre Merzky, Henri Bal
Vrije Universiteit
Amsterdam, The Netherlands
{kielmann,merzky,bal}@cs.vu.nl
Francoise Baude, Denis Caromel, Fabrice Huet
INRIA, I3S-CNRS, UNSA
Sophia Antipolis France
{fbaude,dcaromel,fhuet}@sophia.inria.fr

1 Introduction

A grid, based on current technology, can be considered as a distributed system for which heterogeneity, wide-area distribution, security and trust requirements, failure probability, as well as high latency and low bandwidth of communication links are exacerbated. Different grid middleware systems have been built, such as the Globus toolkit [9], EGEE LCG and g-lite [3], ARC [8], Condor [5], Unicore [4], etc). All these systems provide similar grid services, and a convergence is in progress. As the GGF [6] definition of grid services tries to become compliant to Web services technologies, a planetary-scale grid system may emerge, although this is not yet the case. We thus consider a grid a federation of different heterogeneous systems, rather than a virtually homogeneous distributed system.

In order to build and program applications for such federations of systems, (and likewise application frameworks such as problem solving environments or “virtual labs”), there is a strong need for solid high-level middleware, directly interfacing application codes. Equivalently, we may call such middleware a grid programming environment. Indeed, grid applications require the middleware to provide them with access to services and resources, in some simple way. Accordingly, the middleware should implement this access in a way that hides heterogeneity, failures, and performance of the federation of resources and associated lower-level services they may offer. The challenge for the middleware is to provide applications with APIs that make applications more or less grid unaware (i.e. the grid becomes invisible).

Having several years of experience designing and building such middleware, we analyze our systems, aiming at a generalization of their APIs and architecture that will finally make them suitable for addressing the challenges and properties of future grid application programming environments. In this paper, we identify functional and non-functional properties for future grid programming environments, we present our systems, ProActive, Ibis, and GAT, and investigate which of the properties they meet already. Then we derive a generalized architecture for future grid programming environments, and outline directions of future work.

[*]Full paper published as CoreGRID Technical Report TR-0003, June 2005, http://www.coregrid.net

2 Properties for grid application programming environments

Grid application programming environments provide both application programming interfaces (APIs) and runtime environments implementing these interfaces, allowing application codes to run in a grid environment. Here we outline the properties of such programming environments.

2.1 Non-functional properties

We emphasize non-functional properties as these are determining the constraints on grid API functionality and have to be taken into account when designing grid application programming environments.

- Performance
  As high-performance computing is one of the driving forces behind grids, performance is the most prominent, non-functional property of the operations that implement the functional properties as outlined below.

- Fault tolerance
  Most operations of a grid API involve communication with physically remote peers, services, and resources. Because of this remoteness, the instabilities of network (Internet) communication, the fact that sites may fail or become unreacheable, and the administrative site autonomy, various error conditions arise.

- Security and trust
  A grid API thus needs to support mutual authentication of users and resources. Access control to resources (authorization) becomes another source of transient errors that runtime systems and their APIs have to handle. Besides, privacy becomes important in Internet-based systems which can be ensured using encryption.

- Platform independence
  It is an important property for programming environments to keep the application code independent from details of the grid platform.

2.2 Functional properties

We envision the following categories of necessary functionality.

- Access to compute resources, job spawning and scheduling
  A job submission API has to take descriptions of the job and of suitable compute resources. The mapping and scheduling decisions are usually taken by an external resource broker service [7].

- Access to file and data resources
  Any real-world application has to process some form of input data, be it files, data bases, or streams generated by devices like radio telescopes.

- Communication between parallel and distributed processes
  Besides access to data files, the processes of a parallel application need to communicate with each other to perform their tasks.

- Application monitoring and steering
  Users need to inspect and possibly modify the status of their long-running applications while they are running on some nodes in a grid. For this purpose, monitoring and steering interfaces have to be provided.

3 Existing grid programming environments

For our grid application programming environments, ProActive, Ibis, and GAT, we briefly sketch their scope and how far they address the properties identified above.

3.1 ProActive

ProActive[2] is a Java library for parallel, distributed and concurrent computing, also featuring mobility and security in a uniform framework. With a reduced set of simple primitives, ProActive provides a comprehensive API masking the specific underlying tools and protocols used, and allowing to simplify the programming of applications that are distributed on a LAN, on a cluster of PCs, or on Internet Grids. The library is based on an active object pattern, on top of which a component-oriented view is provided.

3.2 Ibis

The Ibis Grid programming environment [10] has been developed to provide parallel applications with highly efficient communication API's. Ibis is based on the Java programming language and environment, using the "write once, run anywhere" property of Java to achieve portability across a wide range of Grid platforms. Ibis aims at Grid-unaware applications. As such, it provides rather high-level communication API's that hide Grid properties and fit into Java's object model.

3.3 GAT

The Grid Application Toolkit (GAT) [1] aims to enable scientific applications in grid environments. It helps to integrate grid capabilities in application programs, by providing a simple and *stable* API with well known API paradigms (e.g. POSIX like file access), interfacing to grid resources and services, abstracting details of underlying grid middleware. This allows to interface to different versions or implementations of grid middleware without any code change in the application.

3.4 Summary

Table 1 summarizes the functional and non-functional properties addressed by the three systems. There, bullet points indicate properties that are addressed while hollow circles refer to unaddressed properties. From the table it becomes obvious that the three systems have been designed for somewhat different purposes. These choices directly influence which properties are addressed, actually.

Property	**ProActive**	**Ibis**	**GAT**
Non-Functional Properties			
performance	●	●	○
fault tolerance	●	●	●
security / trust	● / ○	● / ○	● / ○
platform independence	●	●	●
Functional Properties			
resources / job spawning / scheduling	● / ● / ○	○ / ○ / ○	● / ● / ●
files / data resources	○ / ○	○ / ○	● / ●
parallel / distributed communication	● / ●	● / ●	○ / ●
application monitoring / steering	● / ○	○ / ○	● / ●

Table 1: Comparison of the three frameworks

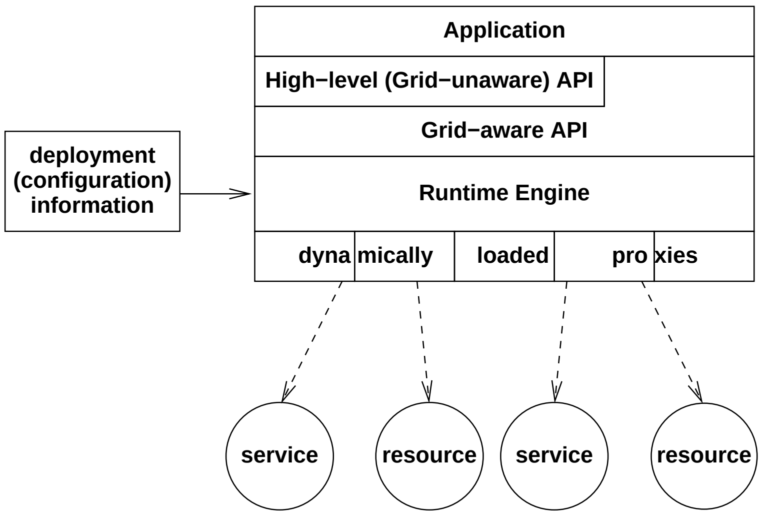

Application
High−level (Grid−unaware) API
Grid−aware API
deployment (configuration) information
Runtime Engine
dyna mically loaded pro xies
service
resource
service
resource

Figure 1: Generic runtime architecture model

4 Generic architecture model

Our three systems, ProActive, Ibis, and GAT, provide API functionality that partially overlaps, and partially complements each other. A much stronger similarity, however, can be observed from their software architectures, which is due to the non-functional properties of platform independence, performance, and fault tolerance. These properties strongly call for systems that are able to dynamically adjust themselves to the actual grid environment underneath.

Figure 1 shows the generic architecture for grid application programming environments that can address the properties identified in Section 2.

- Application code is programmed exclusively using the API's provided by the envisioned grid application programming environment.
- The API's are implemented by a *runtime engine*. The engine's most important task is to delegate API invocations to the right service or resource.
- Delegation to a selected service can be achieved by dynamically loaded proxies.
- For resource and service selection purposes, the runtime engine needs configuration information to provide the right bindings.

It is obvious that current grid application programming environments comply only partially to this architecture. However, with the advent of more sophisticated grid middleware, like grid component architectures, or widely deployed monitoring and information services, also programming environments will be able to benefit and provide more flexible, better performing, and failure-resilient services to applications.

5 Conclusions and future directions

Grids can be considered as distributed systems for which heterogeneity, wide-area distribution, security and trust requirements, failure probability, as well as latency and bandwidth of communication networks are exacerbated. Such platforms currently challenge application programmers and users. Tackling these challenges calls for significantly advanced application programming environments.

We have identified a set of functional and non-functional properties of such future application programming environments. Based on this set, we have analyzed existing environments, emphasizing ProActive, Ibis, and GAT, which have been developed by the authors and their colleagues, which we also consider to be among the currently most advanced systems.

Our analysis has shown that none of our systems curently addresses all properties. This is mostly due to the different application scenarios for which our systems have been developed. Based on our analysis, we have identified a generic architecture for future grid programming environments that allows building systems that will be capable of addressing the complete set of properties, and will thus be able to overcome today's problems and challenges. The full paper provides detailed descriptions and discussions, of the set of properties, the three programming environments, as well as the proposed, generic architecture model for grid application programming environments.

References

[1] Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, Andre Merzky, Rob van Nieuwpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schütt, Ed Seidel, and Brygg Ullmer. The Grid Application Toolkit: Towards Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, 2005.

[2] F. Baude, D. Caromel, and M. Morel. From distributed objects to hierarchical grid components. In *DOA*, volume 2888, pages 1226–1242. LNCS, 2003.

[3] R. Berlich, M. Kunze, and K. Schwarz. Grid Computing in Europe: From Research to Deployment. *CRPIT series, Proceedings of the Australasian Workshop on Grid Computing and e-Research (AusGrid 2005)*, 44, Jan. 2005.

[4] Dietmar Erwin, editor. *Joint Project Report for the BMBF Project UNICORE Plus.* UNICORE Forum e.V., 2003.

[5] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-g: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, 2002.

[6] The Global Grid Forum (GGF). http://www.gridforum.org/.

[7] Jennifer M. Schopf, Jarek Nabrzyski, and Jan Weglarz, editors. *Grid resource management: state of the art and future trends.* Kluwer, 2004.

[8] O. Smirnova, P. Eerola, T. Ekelof, M. Elbert, J.R. Hansen, A. Konstantinov, B. Konya, J.L. Nielsen, F. Ould-Saada, and A. Waananen. The NorduGrid Architecture and Middleware for Scientific Applications. In *ICCS 2003*, number 2657 in LNCS. Springer-Verlag, 2003.

[9] The Globus Alliance. http://www.globus.org/.

[10] Rob V. van Nieuwpoort, Jason Maassen, Gosia Wrzesinska, Rutger Hofman, Ceriel Jacobs, Thilo Kielmann, and Henri E. Bal. Ibis: a Flexible and Efficient Java-based Grid Programming Environment. *Concurrency and Computation: Practice and Experience*, 17(7–8):1079–1107, 2005.