An Integration of Global and Enterprise Grid Computing: Gridbus Broker and Xgrid Perspective

Marcos Dias de Assunção[1,2], Krishna Nadiminti[1], Srikumar Venugopal[1], Tianchi Ma[1], and Rajkumar Buyya[1, 2]

[1]Grid Computing and Distributed Systems Laboratory and [2]NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Parkville Campus, Melbourne, VIC 3010, Australia
{marcosd,kna,srikumar,tcma,raj}@cs.mu.oz.au
http://www.gridbus.org

Abstract. In this work, we present two perspectives of Grid computing by using two different Grid middleware as examples: an Enterprise Grid using Xgrid and a Global Grid with Gridbus. We also present the integration of Enterprise and Global Grids by proposing the integration of Gridbus Broker with diverse enterprise Grid middleware including Xgrid, PBS, Condor and SGE. The sample performance results demonstrate the usefulness of this integration effort.

1 Introduction

Improvements in communication and computing technology have led to the possibility of aggregating diverse, globally distributed, computing and storage resources to form what is now commonly known as Grid [1]. In order to provide users with such a seamless computing environment as the Grid, the middleware for Grid systems need to solve several challenges originating from the inherent features of the Grid, such as heterogeneity [21].

Grids can be classified in two ways, according to their architecture and presence. Considering their presence, we can define two main categories: global Grids and enterprise Grids. These two categories have varying characteristics and are suitable for different scenarios. Global Grids are established over the public Internet, are characterized by a global presence, comprise of highly heterogeneous resources, present more sophisticated security mechanisms, focus on single sign-on and are mostly batch-job oriented [10]. Enterprise Grids consist of resources spread across an enterprise and provide services to users within that enterprise and are managed by a single organization [12]. They can be deployed within large corporations that have a global presence though they are limited to a single enterprise [13]. Such Grids are more concerned with cycle stealing from unused desktops within enterprises and security mechanism design is not as difficult as it is for global Grids since they are mostly established within the borders of a single organization [11].

Although these Grids present different functions and abilities, organizations are moving towards using both the heterogeneous computing and storage capabilities of

global Grids along with the utility maximization offered by enterprise Grids. For example, a company may want to fetch data from a data repository shared by academic and enterprises but leverage its infrastructure by processing the data on its own enterprise Grid. Organizations may also want to go beyond their Grids to share resources with new partners when their applications require computing resources that surpass what their own Grids can offer [12]. In this way, by using extra resources offered by other partners they can improve their performance as well as increasing their agility. We need middleware to integrate global and enterprise Grids in order to enable such scenario.

Integration of global and enterprise Grid middleware is motivated by offering a uniform API that helps users to write applications for all kind of Grid middleware. In this paper, we present how this was achieved by extending Gridbus Broker [2] to support four different middlewares: Xgrid [4], PBS [15], Condor [14] and SGE [16]. Xgrid is a Grid technology that provides means to build Grids of Mac OS X based computers. PBS is a batch queuing system that provides controls over initiating or scheduling execution of batch jobs. SGE provides distributed resource management software for wide ranging requirements from compute farms to Grid computing. Similarly, Condor supports high throughput computing on collections of distributed computing resources. As a case study, we present the process of integrating Xgrid with Gridbus Broker in detail.

The remainder of this paper is organized as follows. Section 2 presents a discussion about middleware and the positioning of the broker in integrating enterprise and global Grids. We also discuss in detail the characteristics of the Apple's Xgrid technology and justify why we chose to integrate the broker with this middleware. We demonstrate how we have been developing interfaces to different middleware in Section 3. In Section 4, we present how to implement an interface between the Gridbus Broker and Xgrid as a case study. Section 5 presents some experiments demonstrating the usability of the interface. We describe related work in Section 6. Finally, Section 7 presents future work and concludes the paper.

2 Grid Middleware and Background Technologies

During the last few years, research communities have proposed and developed several middlewares for Grid computing that each address issues such as security, uniform access, dynamic discovery, aggregation and quality-of-service differently. Although there have been efforts of standardizing various middleware interfaces and functions [22], they still present some distinguishing characteristics that need to be tackled individually. A layered view for Grid architecture is commonly adopted [21].

The *Fabric* layer provides resources for which the shared access is mediated. Components of the fabric layer interface provide specific functions and services that are used by operations in the upper layers. *Low-level* middleware offers services such as remote process management, co-allocation of resources, storage access, information registration and discovery and security. These services abstract the complexity and heterogeneity of the fabric level by providing a consistent method for accessing

distributed resources [5]. *User-level* middleware uses the uniform interfaces provided by the low-level middleware to provide higher-level abstractions and services [6]. Several *applications* have been developed on top of these layers by using Grid-enabled languages and utilities.

Nowadays, Grid technologies are available for UNIX based OSes and Windows, while Xgrid is an intent to construct Grids of Mac computers. Moreover, there is a great intent in integrating such enterprise Grid technologies with global Grids so that enterprises can maximize their utility and tap into resources across several organizations. The user level middleware and tools such as a broker play an important role in this scenario. Considering these factors, we can perceive that providing a broker for these technologies, that facilitates the aggregation, selection and scheduling of applications in different technologies, is essential.

The Gridbus Broker supports computational and data Grid applications [2] and its architecture has an emphasis on simplicity, extensibility and platform independence. The main design principles of the broker include:

Assume nothing about the environment: It is just assumed that low-level middleware is able to provide an interface to submit and monitor a job.

Client-centric design: the broker does not depend on metrics provided by the resources. No software, besides the middleware itself, needs to be installed on resource side.

Extensibility is the key: the broker is extensible in many ways. We can implement support for new middleware; new information sources can be added; and the XML-based language used to specify jobs and resources is highly extensible.

2.1 The Xgrid Architecture

Xgrid's architecture is similar to other desktop middleware systems such as Condor, and consists of the following components: agents, clients and the controller [9]. In the normal flow of execution, clients originate and submit jobs to the controller; they are split into tasks by the controller and sent on to agents. The agents execute tasks and return results to the controller, which collects them and reports to the client.

Some features provided by Xgrid include [8]: (a) easy Grid configuration and deployment; (b) straightforward and flexible job submission; (c) Kerberos single sign-on and password based authentication; (d) hides complex issues such as data distribution, job execution, and result aggregation from the user; (e) uses open standards; and (f) provides tools for the customization of the job submission process;

Some ongoing research projects are currently using Xgrid. For example, the Xgrid at Stanford is a project from Stanford University [7] and aims at harnessing processing cycles from computers from all over the world. The purpose is to *modelize* the conformational changes of the beta 2 adrenergic receptor, and have a better understanding of its pharmacology.

3 Interface to Different Enterprise and Global Grid Middleware

In order to integrate a different middleware into the broker, an actuator specific to that middleware needs to be implemented; it is responsible for dispatching and monitoring the job. This is done by extending two classes: `ComputeServer` and `JobWrapper`. The implementation of the `JobWrapper` class implements methods necessary for job submission to a resource by using the corresponding middleware, while `ComputeServer` provides means to query the job execution status and discover server's properties.

Interfaces to diverse middleware were implemented, and are summarized in Table 1. Currently, we have implemented adaptors for heterogeneous middleware systems, namely Fork (for forking jobs on local UNIX-like systems), Xgrid, PBS (Portable Batch System), SGE (Sun N1 Grid Engine) and Condor. As we know, PBS, SGE and Condor are all technically mature and widely-adopted computational management middleware for clusters and LAN-based distributed systems. They can link nodes to follow their particular structure so that optimizations could be carried on the managed domain, in order to achieve respective goals. The broker will abstract these irregular structures into general computational resources, while keeping the benefit and autonomy of each middleware.

Conceptually, the middleware implementation works in three "spaces" (Fig. 1). A space is defined as the execution environment, including paths and user accounts. The user space is where users start the broker, as well as the source of all input files and the destination of all output files. The driver working space is a client node (or head node) of the target middleware. The broker provides a dispatcher to control remotely the data and behaviors in the driver working space. The dispatchers also provide the functionality of staging files between the two spaces. The dispatchers are based on some remote logon channels. Currently we have implemented local (meaning the broker itself sits in the driver working space) and SSH (Secure Shell) dispatchers. There exists another space called the middleware inner working space, which is maintained by the middleware systems, on their executing nodes. Also the middleware systems have their own mechanism to stage files and data between the inner working space and their client nodes (that is, the driver working space).

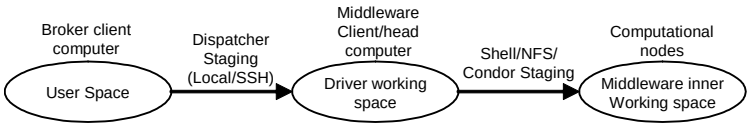

Fig. 1. The three spaces.

There is also similar mechanism provided by the Globus Toolkit [20] (by implementing a driver to incorporate middleware with the GRAM service). As stated earlier, Gridbus broker has been designed to support coordinated use of Grid resources that are accessible via Globus or other services. The Gridbus broker aims to provide an environment that scales and supports full utilization of all types of local and remote resources (desktops, supercomputers and clusters). Therefore, we have extended our Gridbus broker so that it can support scheduling of applications on local and remote resources irrespective their access interfaces (e.g., direct access using local interfaces, SSH-based remote access, or PKI/Globus-based access). This is especially

useful in case of Xgrid as to the best of our knowledge there is no Globus-based access to Xgrid resources. Thus, our integration solution supports uniform and simultaneous use of local and remote resources regardless their access interfaces or mechanisms.

Table 1. Characteristics of the interfaces for different middleware.

Middl.	File Stage-In	File Stage-Out	Execute Job	Query Job Status
Fork	Use symbolic links to avoid an additional copy	Move the output files to the specified directory	Encapsulate the execution and process ID reflection commands into a shell script, then directly execute it, get the process ID from the script's standard output.	Analyze "ps -a" command and obtain the status
Condor	Similar to Fork for NFS. Otherwise, specify the files to-be-staged-in in the condor script	Similar to Fork for NFS. Otherwise, specify the files to-be-staged-out in the condor script	Encapsulate the submission of the condor script into another shell script. Also the shell script is responsible for analyzing the output of the "condor_submit" command while reflecting the condor PID	Analyze the output of the "condor_q" command (executed as a shell script)
SGE	Same as Fork in an NFS system.	Same as Fork in an NFS system.	Explore the suitable queue using a Fork job. Then customize a SGE script for submitting job into the best queue. Another script is responsible for executing the "qsub" command to submit the SGE script and get the SGE PID	Analyze the output of the "qstat" command (executed as a shell script)
PBS	Same as Fork in an NFS system.	Same as Fork in an NFS system.	Find best queue by using a Fork job. Then customize a PBS script for submitting job to the best queue. Another script is executes the "qsub" command to submit the PBS script as well as getting the PBS PID	Analyze the output of the "qstat" command (executed as a shell script)
Xgrid	Copy files to Xgrid input directory.	Move the output files to the specified directory.	Uses Xgrid command line to submit a job. Xgrid ID is used to query job status, redirect output files and delete the job after it was been completed.	Analyze the "xgrid –job status/attributes" command.

Fig. 2 shows a state machine indicating the flow for the broker dispatching jobs to a middleware system. The main steps include file stage-in/out, job execution and job status query. In most cases, we generate a new shell script to interact with the middleware. For different middleware, the driver can adapt to local shell command interfaces. The next sections show how this happens for each middleware.

For the Fork adaptor, it creates symbolic links for stage-in and moves files for stage-out. The driver creates a new process calling the shell script to do the file staging, then execute the job and redirect the stdout and stderr streams to specified

files and stages these files back right after their creation. The process ID of the job is reflected in the stdout stream, which is parsed by the broker. The process ID is the job handle. After the job submission, the broker queries job status by analyzing the output of a "ps –a" (for listing all the processes in UNIX-like systems) command. If the job is active or pending, its status can be obtained from the "ps" output, otherwise, the broker parses the stderr stream to determine whether the job has been successfully finished (if there is nothing in the stderr) or failed. If the job is finished, the output files generated by the job are staged back to the broker side.

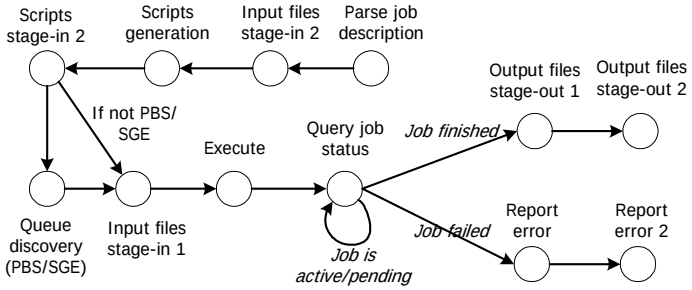

Fig. 2. State machine of the driver flow. For Stage-In and Stage-Out: 1 represents between the user space and driver working space. 2 represents between the driver working space and middleware inner working space.

For PBS and SGE driver, it will not stage the input files between the driver working space and the middleware inner working space, because they are configured to share the same NFS. Nevertheless, the execution scripts describing the job configuration need to be staged to the correct directory specified by the middleware and all file attributes have to be updated so that the files could be accessible under the middleware execution environments. In addition, after the execution, the location of output files (also specified by the middleware) is given to the dispatcher. In PBS/SGE, the computational resources are further partitioned into queues. In our current implementation, the user can either specify the queue(s) to be used by the job submission or let the broker select the most suitable one from all the available queues. In the second case, the broker submits a queue discovery job (marked as a Fork job) prior to the PBS/SGE job in order to select the "best queue". Then the PBS/SGE job is submitted to the specified or selected queue by another job submission shell script by running "qsub" (the job submission interface) to submit the execution script to PBS/SGE. The job handle is set as the PBS PID or SGE PID, which is retrieved from the output of the job submission script. The broker queries the job status by running the command "qstat" through a shell script and analyzing the stdout stream.

With the Condor driver, the staging of both the input and output files is handled automatically by Condor. The adaptor needs to specify all the files in the Condor execution script (needed by Condor for describing the job configuration). Then the adaptor generates a shell script to submit the execution script to Condor and retrieves the Condor PID, which is the job handle for Condor jobs. After the job submission, the broker queries the job status using a shell script by running "condor_q". The *standard* Condor universe provides the feature of checkpointing and process

migration. However, as a requirement, the job must be re-linked to the Condor library using “condor_compile”, which is unpractical for most jobs without the source code. Therefore, the broker submits the job into a universe called “vanilla”.

4 Design and Implementation for Xgrid

As discussed beforehand, the scheduler in Gridbus broker does not assume anything about the diverse middleware that may be utilized by users. Two classes have to be implemented so that the broker is able to submit jobs to different middleware, which are `ComputeServer` and `JobWrapper` as described in Fig. 3.

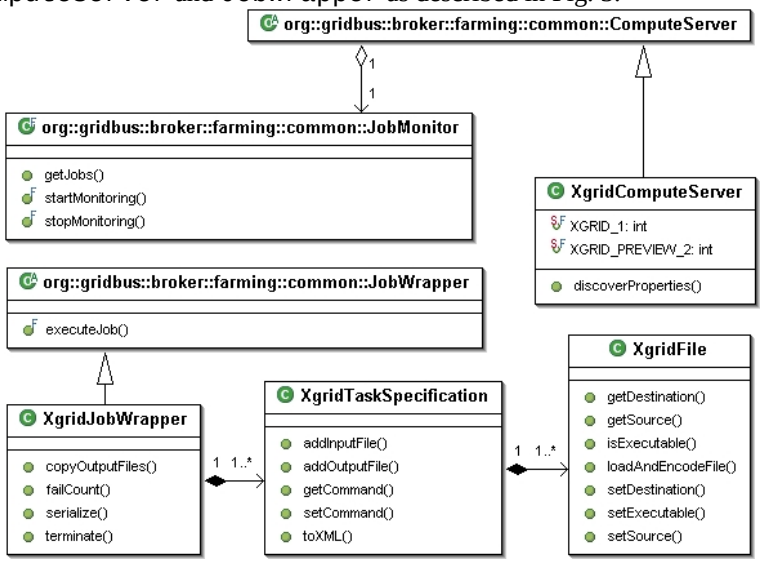

Fig. 3. Class diagram of the Xgrid adaptor.

Gridbus Broker aims at being platform independent and client-centric. Following this approach, we assumed that the broker can be installed on a computer under a different operating system from Mac OS X, while it submits jobs to Xgrid. To the best of our knowledge, at the present there is no Xgrid API for Windows or UNIX class operating systems. Therefore, we proceed according to the following steps:

The user performs the job submission using Gridbus Broker informing it of the scripts or commands to be executed on the remote node as well as the necessary input files.

The submission is carried out using the command line application on an Xgrid client. Since Xgrid’s local working directory is specified, and all files contained in this directory will be compressed into a file that is sent to the controller. The results will be available in the local working directory when the job execution is finished. The broker node can use this node directly if it is being run natively on the same node, or via SSH when it is running on a remote node and possibly under a

different operating system.

Once the job completes in Xgrid, the broker copies the files back from the Xgrid client node to the broker node.

A ComputeServer in the broker represents a resource. To implement an interface to a new middleware one has to extend this class and implement the methods responsible for discovering properties of the resource and for querying the job status, by using the tools specific to the corresponding middleware. For Xgrid, a ComputeServer represents a cluster of Mac computers.

While each middleware has its own representation of a job, the broker uses its own generic representation. The XgridJobWrapper class converts the job description from the broker representation to the Xgrid specific representation and submits it. In the broker, a job can consist of tasks such as SUBSTITUTE, COPY and EXECUTE. The copy tasks may represent input files that are required by the job or files that have to be copied back after the job has been completed.

The Xgrid Version 1.0 accepts XML documents containing serialized input files, description of commands to be executed, arguments. Since the interface was implemented to work for both Technical Preview 2 and 1.0 versions of Xgrid, the input files are copied as a single XML document for version 1.0, and as separate files for the Technical Preview 2 version. When the job is submitted, a job ID is obtained from Xgrid, which is used to monitor the job execution.

5 Performance Evaluation

To evaluate the interfaces to Xgrid, we conducted several experiments. This section describes results of experiments we carried out for (a) Xgrid and (b) combined use of SGE and PBS resources.

5.1 The Xgrid Testbed and Results

For the evaluation of the integration of Xgrid and its usage via Gridbus Broker, we have used a cluster composed of Mac OS X computers located in the Howard Florey Institute for Neuroscience at the University of Melbourne. This cluster has 13 nodes that are connected by a Fast Ethernet LAN. Each node has two Power PC processors of 1.8GHz. The Xgrid Technical Preview 2 is installed on this cluster, which is a beta version of Xgrid. Since version 1.0 of Xgrid requires an upgrade of the operating system on all nodes to version 10.4 of Mac OS X and the computers are used for other purposes such as mail and web servers, we are unable to install Xgrid version 1.0 on this cluster at the moment. For future experiments, we aim at using version 1.0, which is a stable production release.

For validating our integration with Xgrid, we have used a sample of application for the jobs submitted to our testbed. The jobs consist of an executable file that is copied to the Xgrid client node. This application generates 16,000 digits for PI and takes about 12 seconds to be executed on the head node of the Xgrid cluster. This application creates an output file of 16KB with the digits and no stdout. In this way,

the files that are copied back are this output file, the standard output that is of size 0 and stderr that will be of size 0 in case of success.

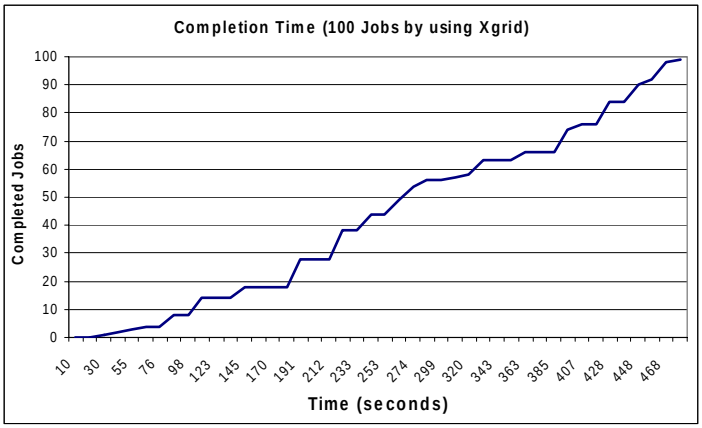

Fig. 4. Execution time (100 jobs by using Xgrid TP2).

The scenario evaluated with Xgrid was the execution of a parameter-sweep application of 100 jobs described using XPML (The XML-based parametric modeling language supported by the Gridbus Broker). During the execution, we measured how many jobs were completed over the execution time. Fig. 4 presents the results of this evaluation. In this experiment, 99 out of 100 jobs were successfully completed. For some unknown reason, the Xgrid command line application hangs sometimes. Hence, it results in an increase of the time the job takes to execute.

We evaluated the round trip time for job submission for Xgrid when using SSH from a computer in the same local area network. The round trip time can present some variation since often users have been utilizing the network for other purposes, and according to these results, all the jobs will have increase in the execution time of about 1 second in the testbed's local area network.

5.2 The PBS/SGE Testbed and Results

We have used two clusters in this evaluation. The first one, the same as the above, is composed of Mac OS X machines located within the Howard Florey Institute for Neuroscience at the University of Melbourne, but SGE was used to manage its resources. The second cluster, called Manjra, is composed of 12 Linux nodes with 2.40GHz Intel Pentium 4 CPUs and 512MB of memory and it is located in another building (GRIDS Lab) at the University of Melbourne. We used PBS to manage Manjra's resources. The job submission to PBS was done using SSH/SCP and the results are presented in Fig. 5. Since the second cluster is running a different operating system (Linux) and as the executables are not portable between Mac OS X operating system and Linux, we implemented the PI calculator described previously as a Java application. On the head computer in the cluster managed by SGE this application takes around 3-5 minutes to finish its execution by using Java 1.4.

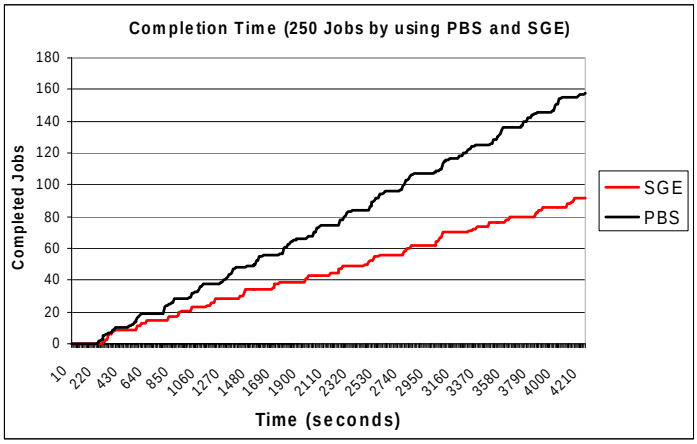

Fig. 5. Execution time (250 jobs by using SGE and PBS).

As presented in the last graph, the interface for PBS presented a better performance compared to SGE. PBS executed 158 out of 250 jobs, whilst SGE executed 92 jobs. The Xgrid experiments were carried out by using an executable application for Mac OS X. We are satisfied with the results even though this release presents some issues such as failing of job submission. We plan to carry out experiments by utilizing the version 1.0.

6 Related Work

There exist some works aiming at integrating the middleware described in this paper with other technologies. For instance, pools of resources managed by Condor can be integrated with other Condors pools by setting up the manager of a cluster to accept requests from another manager. Such an approach is commonly called a flock of Condors [17]. In this case, a pool *B* will send jobs to a pool *A* if the local resources are unavailable or in use.

TrellisWeb [18] provides an interface to a placeholder scheduling that addresses resource scheduling and allocation, single log on and access control. The system is integrated with PBS, SGE and IBM's LoadLeveler and applies these tools in a metacomputing scenario. However, this work does not contemplate integration with global Grid middleware such as Globus and Unicore.

Grid(Lab) Grid Application Toolkit (GAT) [19] presents an architecture in which common APIs sit between Grid applications and diverse Grid middleware. The main goal of GAT is providing Grid programmers with a uniform interface to different middleware and services, such as data catalog, data replication, data movement, Globus Grid Services and Globus 2.X and 3.X Pre WS. By providing such an API, GAT aims at enabling the easy development of "Grid-aware" applications. This API targets the development of portals and can therefore enable a range of Grid applications. Although this project shares many characteristics with our approach, it is

not targeted towards at integrating the API with enterprise middleware.

Our integration differs still from the works mentioned above in that Gridbus broker does not provide the integration with only one toolkit, seen with the flock of Condors, but instead integrates several enterprise and global Grid middleware. Furthermore, the Gridbus Broker provides a uniform interface that users can utilize to specify their parameter sweep applications by using an XML-based parametric modeling language as well as Grid portals. The scheduler in Gridbus Broker is middleware independent and can be set to optimize user-supplied parameters such as deadline and budget.

7 Conclusions and Future Works

Gridbus project provides a broad range of tools and aims at leveraging Grid technologies for different platforms. Through its broker, Gridbus targets to provide a common interface and API for users to develop their applications for enterprise and global Grids.

We have developed interfaces to integrate different middleware with the Gridbus Broker. By integrating such technologies with the broker, we aim at leveraging global Grids. Users can use their Macs as well as other computers via integration with other systems.

We also present some experiments that demonstrate the usability of the interface for Xgrid, SGE and PBS. Our experiments consisted of running a parameter sweep application. A description about how to implement interfaces for other middleware was provided along with the results of the experiments that were carried out.

As future work, we propose to adapt the Gridbus interface to Xgrid make use of the BEEP protocol to communicate directly with the Xgrid controller, avoiding using the Xgrid command line application to submit jobs and monitor results. By doing so, we can collect more information about nodes in the Grid and the jobs submitted to Xgrid. We also aim at carrying out tests for the interfaces for other middleware described in this paper.

References

1. Ian Foster, Carl Kesselman, and Steve Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, 15(3), 2001.
2. Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A grid service broker for scheduling distributed data-oriented applications on global grids. In *Proceedings of the 2nd workshop on Middleware for grid computing*, pages 75–80, New York, USA, 2004.
3. Jim Almond and Dave Snelling. Unicore: uniform access to supercomputing as an element of electronic commerce. *Future Gener. Comput. Syst.*, 15(5-6):539–548, 1999.
4. David A. Kramer and Matthew MacInnis. Utilization of a local grid of mac os x-based computers using xgrid. In *HPDC*, pages 264–265. IEEE Computer Society, 2004.
5. Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal. *High Performance Computing: Paradigm and Infrastructure*, chapter Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies. ISBN: 0-

471-65471-X. Wiley Press, New Jersey, USA, June 2005.
6. Rajkumar Buyya and Srikumar Venugopal. The gridbus toolkit for service oriented grid and utility computing: An overview and status report. In *Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004)*, ISBN 0-7803-8525-X, pages 19–36, Seoul, Korea, April 2004. IEEE Press.
7. Blane Warrene. Stanford University Lab builds an xgrid. Macnews: Hardware, Aug. 2004.
8. Stuart Bowness. Xgrid and cross platform grid computing. 440 project report, University College of the Fraser Valley, April 2005.
9. Xgrid: High performance computing for the rest of us. Apple Developer Connection, March 2005.
10. Ahmar Abbas. *Grid Computing: A Practical Guide to Technology and Applications*. Charles River Media, Hingham, Massachusetts, first edition edition, 2004. ISBN: 1-58450-276-2.
11. Geoffrey Fox and David Walker. e-science gap analysis. UK e-Science Technical Report Series, June 2003.
12. Enterprise grid alliance reference model v1.0. Enterprise Grid Alliance, April 2005.
13. Ian Baird. Grids in practice: a platform perspective. MIDDLEWARESpectra, June 2003. www.middlewarespectra.com/grid.
14. The condor project homepage. http://www.cs.wisc.edu/condor/.
15. Openpbs: The portable batch system software. Veridian Systems, Inc., Mountain View.
16. Sun grid engine (sge): A cluster resource manager. http://gridengine.sunsource.net/.
17. Ali Raza Butt, Rongmei Zhang, and Y. Charlie Hu. A self-organizing flock of condors. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 42, Washington, DC, USA, 2003. IEEE Computer Society.
18. Christopher Pinchak, Paul Lu, Jonathan Schaeffer, and Mark Goldenberg. The canadian internetworked scientific supercomputer. In *17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS)*, pages 193–199, Sherbrooke, Quebec, Canada, May 2003.
19. Gabrielle Allen, Kelly Davis, Tom Goodale, Andrei Hutanu, Hartmut Kaiser, Thilo Kielmann, AndreMerzky, Rob Van Niewpoort, Alexander Reinefeld, Florian Schintke, Thorsten Schutt, Ed Seidel, and Brygg Ullmer. The Grid Application Toolkit: Toward Generic and Easy Application Programming Interfaces for the Grid. *Proceedings of the IEEE*, 93(3):534–550, March 2005.
20. Ian Foster and Carl Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.
21. Mark Baker, Rajkumar Buyya, and Domenico Laforenza. Grids and grid technologies for wide-area distributed computing. *International Journal of Software: Practice and Experience (SPE)*, 32(15):1437–1466, December 2002.
22. Global Grid Forum. http://www.ggf.org, 2005.