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

Marcos Dias de Assunção, Krishna Nadiminti, Srikumar Venugopal, Tianchi Ma, Rajkumar Buyya

Grid Computing and Distributed Systems Laboratory
Dept. of Computer Science and Software Engineering
The University of Melbourne, Australia
Email: {marcosd,kna,srikumar,tcma,raj}@cs.mu.oz.au

**Abstract**

Several middleware for grid computing have been proposed aiming at providing means to access resources in a uniform and secure way, hiding the idiosyncrasies of heterogeneous resources. In this work we present the Enterprise Grids from the Xgrid perspective and the Global Grids perspective from Gridbus. We also present the integration of Enterprise and Global Grids by proposing the integration of Gridbus Broker with diverse enterprise grid middleware technologies including Xgrid, PBS, Condor and SGE. The implementation of Xgrid Actuator that enables the use of Xgrid services via Gridbus Broker is presented as a representative implementation. 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 popularly called as Grid. In order to provide users with such a seamless computing environment, the middleware for Grid systems need to solve several challenges originating from the inherent features of the Grid [1].

Grids can be classified in two ways, according to their architecture and coverage. Considering their coverage 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 strong security mechanisms since the grid can comprise different administrative domains, focus on single sign-on and are mostly batch-job oriented [14]. Enterprise grids consist of resources spread across an enterprise and provide services to users within that enterprise and are managed by a single organization [17]. They can be deployed within large corporations that have a global presence even though they are limited to a single enterprise [19]. Such category of grids is 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 it is established within the borders of a single company [16].

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 know new partners to share resources when their applications require computing resources that surpass what their grids can offer [17]. In this way, by using extra resources offered by other partners they can improve their performance while increasing their agility. Therefore, we need middleware to integrate global and enterprise grids.

Integration of global and enterprise middleware needs to offer uniform APIs and familiar interfaces for users, such as resource brokers. By providing such common APIs, users can develop their applications for both kinds of grid. In this way, the integration of global and enterprise grids is more likely to be done in user level middleware by tools such as resource and service brokers. In this paper we present how this was achieved by extending Gridbus Broker [2] to support 4 different middleware: Xgrid, PBS, Condor and SGE. We present the integration to Xgrid in detail as a case study. Xgrid [5] is a grid technology from Apple Computer, Inc. that provides means to build grids of Mac OS X based computers.

The remainder of this paper is organized as follows. The 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 we justify why we chose to integrate the broker with this middleware. We

demonstrate how interfaces to different middleware have been done 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. Related work is described in Section 6. Finally, Section 7 presents future work and concludes the paper.

## 2. Grid Middleware and Background Technologies

Middleware is the software layer placed between the operating system and the applications and provides a set of services required by applications. The middleware hide the idiosyncrasies of different operating systems and enables the development of distributed systems providing a uniform and reliable access to resources. Several middleware for grid computing have been proposed and developed over the last few years and they try to address issues such as security, uniform access, dynamic discovery, aggregation and quality-of-service. Although there are great standardization efforts, they still present some distinguishing characteristics.

A layered approach for Grid architecture is presented in Figure 1. The architecture consists of the following layers: fabric, core middleware, user-level middleware, and applications [6].

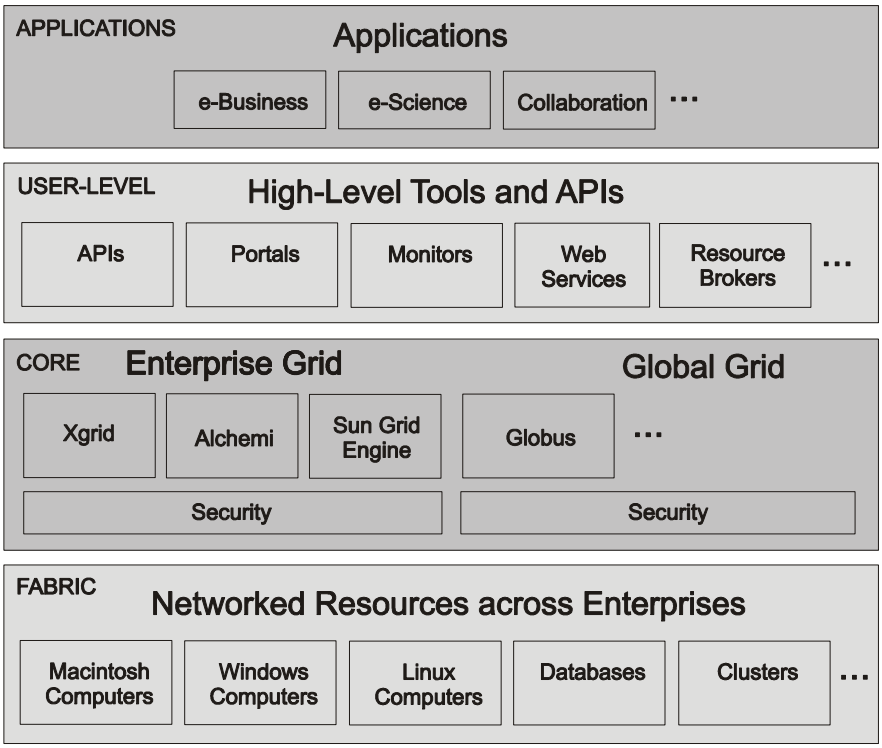


**Figure 1 - Layered view of grids toolkits.**

The *Fabric* layer of grid provides resources for which the shared access is mediated such as computational resources, storage systems, catalogues, network resources and sensors networks. A resource can be a physical or logical entity, as a distributed system, cluster of computers or a group of distributed computers. The components of the fabric layer implement local and specific operations on the resources as a result of the operations of the higher layers.

*Core Grid* middleware offers services such as remote process management, co-allocation of resources, storage access, information registration and discovery, security, and aspects of Quality of Service (QoS) such as resource reservation and trading. These services abstract the complexity and heterogeneity of the fabric level by providing a consistent method for accessing distributed resources [6].

*User-level Grid* middleware uses the uniform interfaces provided by the low-level middleware to provide higher level abstractions and services [7]. The broker is a key component in this level and is used to submit jobs across

both enterprise and global grids technologies.

Several *Grid applications* are constructed on top of grid tools, generally using Grid-enabled languages and utilities such as HPC++ or MPI. Grid portals are also built based on the tools provided by underlying layers. A Grid portal is a web server that provides an interface to Grid services, allowing users to submit compute jobs, transfer files, and query Grid information services from a standard web browser.

The broker has the goal of providing access to resources, hiding the heterogeneity of different middleware. This way, the Gridbus broker was designed to work on top of different middleware. The broker provides access to services provide by both enterprise and global grids [2].

### 2.1. The Gridbus Broker

The Gridbus Broker is designed to support computational and data grid applications [2]. The architecture of the broker has emphasis on simplicity, extensibility and platform independence. It is implemented in Java and provides transparent access to grid nodes running various middleware. The main design principles of the broker include:

- Assume nothing about the environment – It is just assumed that the resource is able to provide an interface to submit a job. The grid does have a close integration with middleware. It is fault-tolerant and faults are dealt by a recovery module.
- Client-centric design – the broker does no depend on metrics specified by resources and no additional software, except the middleware, need be installed on resource side.
- Extensibility is the key – the broker can be extended in several ways. Support for new middleware can be implemented; new information sources can be added; and the language used to specify jobs and resources is highly extensible.

The Gridbus broker's architecture can be described as composed of the following subsystems:

- the application interface sub-system;
- the core-sub-system; and
- the execution sub-system.

The application interface is the entry point to the broker, and its input is an application description that consists of tasks and associated parameters, and a resource description. The application-description is converted into entities called jobs. The tasks and resources description drives the discovery of resources. The resource discovery module verifies if the resources are available and if they are suitable for the current set of jobs. The scheduler maps the jobs to the corresponding servers based on the chosen scheduling algorithm. The actuator is a middleware specific component responsible for dispatching the job. The resources are represented by entities called servers.

In order to integrate a different middleware into the broker, two classes have to be implemented: `ComputeServer` and `JobWrapper` for each specific middleware. The implementation of the `JobWrapper` class has to provide the methods necessary to submit a given job to the corresponding middleware, while the implementation of the `ComputeServer` offers means to query the job's execution status and discover server properties.

Following this approach the broker is currently able to submit jobs to diverse enterprise grid resources powered using middleware such as Alchemi [4], PBS, Sun Grid Engine, Xgrid and global grids such as Unicore [3] and Globus [27]. In this work we present how it has been done for Xgrid so that Mac OS X based computers can be integrated into global grids. Thus, we discuss some characteristics of Xgrid and furthermore we present how the broker works together with different middleware, including Xgrid, and how to plug-in support for other middleware.

### 2.2. The Xgrid Architecture

As in this paper we propose the integration of Gridbus broker with different enterprise middleware and present how it has been done for Xgrid, it is interesting to know more about Xgrid. Xgrid is a distributed computing software technology from Apple Computer, Inc [13], which aims at providing a configuration for high performance computing using Mac OS X based computers. Compared to other grid technologies it has some advantages in enterprise grids [11]:

- Easy grid configuration and deployment.
- Straightforward and flexible job submission.
- Kerberos single sign-on and password based authentication (in the version 1.0). However, Apple recom-

mends that the Kerberos system be used only for grids under the same administration.

- Automatic discovery of controllers by agents and clients.
- Hides complex issues such as data distribution, job execution, and result aggregation from the user.
- Uses open standards (e.g. BEEP protocol [8][9] is used for the communication among the components and the communication can theoretically use SSL).
- Tools for the customization of the job submission process.

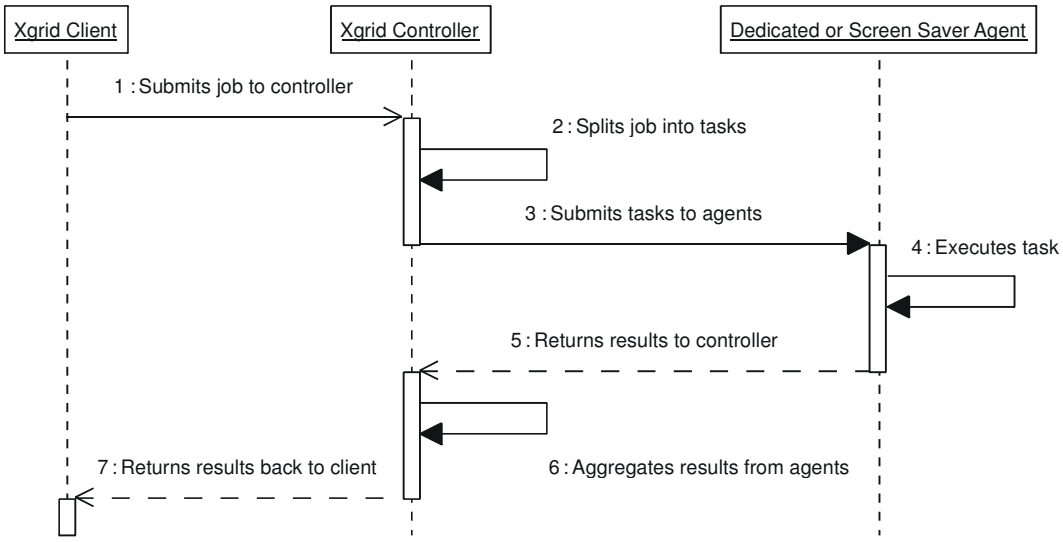


**Figure 2 - Xgrid workflow.**

For some of the aforementioned characteristics we chose to integrate the broker with Xgrid. Moreover, Xgrid is supported by Apple and it specifically target to Mac OS X based computers. If the user has Globus installed in his Mac computers, the broker will work with it. However, if due to some of the reasons mentioned above, such as easy of use and install, the user prefers to adopt Xgrid, he can use the broker since it provides the integration with this middleware.

The architecture of Xgrid is similar to other desktop middleware systems such as Alchemi and Condor, and is basically composed of the following components: agents, clients and the controller. The system provides the user with a command line interface (i.e. a command line tool called xgrid) and also allows the user to implement and run customized plug-ins that can be loaded through a GUI interface. The Xgrid workflow is shown in Figure 2 [12]. The client originates and submits jobs to controller (1); they are split into tasks by the controller (2) and sent on to agents (3). The agents execute tasks (4) and the completed tasks are returned to the controller (5) which collects them (6) and reports back to the client (7).

There are some ongoing research projects that are currently using Xgrid. For example, the Xgrid at Stanford is a project from Stanford University [10] 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.

### 2.2.1. Security Aspects of Xgrid

Xgrid uses single passwords as default. The version 1.0 also provides single sign-on by using Kerberos. In order to use Kerberos, the controller must be installed on a machine running the server version of Mac OS X. When using single passwords, the client must provide the controller with a password if it wants to have a job executed. The controller must provide a password to authenticate with agents and vice-versa. The authentication process is composed of a 2-way random mutual authentication protocol that includes MD5 hashes.

The controller is the only component of Xgrid which has a listening port (i.e. by default the port 4111 is used). The clients establish a connection with the controller to ask for jobs and they do not have any listening port. This

approach enables and simplifies the use of agents and clients behind firewalls.

When using the password based authentication, the jobs in Xgrid run as the user 'nobody' and can execute files allowed to this user. Moreover, the Access Control Lists (ACL) available in Mac OS X v10.4 allow a powerful and flexible way to assign access permissions to users. With the Kerberos authentication different permissions can be set according to each user.

#### 2.2.2. Job Submission in Xgrid

The job submission in Xgrid can be carried out using plug-ins and also by a command line interface. By using the plug-in interface one can specify things like parameter sweep applications and save it for future use. It is also allowed to choose the local directory contained the files to be copied to agents and where the results will be stored after processing the job. With some limitations, the command line tool offers similar things. Once the submission is started, the following actions occur [11]:

- All files contained in the local directory are compressed into a file. In this directory one can put his script or executable file if it is not installed in agent machines.
- The controller stages the compressed file with all the directory structure to the first agent with the first parameter in case of a parameter sweep application.
- Each agent receives a compressed file containing the structure directory and a parameter between the ranges.
- The agent receives the compressed file and extracts it to the /tmp directory, which becomes the agent's working directory.
- The command is executed and the STDOUT is placed in a text file that is sent back and all output files are compressed into a file which is returned to the controller.
- The results are sent to the client and will be available in the local working directory.

The Xgrid currently allows several forms of jobs: Shell basically consists of UNIX commands; A Xfeed job allows one to specify either a command or the path to a script; MPI-based applications in which case the development of applications is possible using the MacMPI library.

## 3. Interface to Different Enterprise and Global Grid Middleware

Interfaces to diverse middleware have been implemented for the Gridbus Broker. We summarize how this has been done for other middleware by describing the characteristics of the interfaces for the broker in Table 1.

Currently we have implemented drivers for four heterogeneous middleware systems (besides Xgrid), namely Fork (for forking jobs on local UNIX-like systems, such as Linux, Solaris and Mac-OS), PBS (Portable Batch System [21]), SGE (Sun N1 Grid Engine [22]) and Condor [20]. 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 (enhanced throughput, scalability or fault tolerance). 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" (Figure 3). 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. It is the node where the broker can land the jobs onto the local administrative domain. The broker provides a dispatcher to remotely control 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 (mean 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).

**Table 1 - Characteristics of the interfaces for different middleware.**

| | File Stage-In | File Stage-Out | Execute | Job Handle | 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 the script, get the process ID from the standard output of the script | The process ID | Analyze the output of the "ps -a" shell command (different usage for each OS) and retrieve the status |
| **Condor** | Same as Fork in an NFS system. For a non-shared FS, specify the files to-be-staged-in in the condor script | Same as Fork in an NFS system For a non-shared FS, 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 | 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 best-suited queue using a Fork job. Then customize a SGE script for submitting job into the best queue. Another shell script is responsible for executing the "qsub" command to submit the SGE script, as well as reflecting the SGE PID | 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. | Explore the best-suited queue using a Fork job. Then customize a PBS script for submitting job into the best queue. Another shell script is responsible for executing the "qsub" command to submit the PBS script, as well as reflecting the PBS PID | 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 interface for job submission. Uses the Xgrid ID to query about job status, redirect output file to destination directory and delete the job after it has been completed. | The Xgrid ID | Analyze the output of "xgrid –job status/attributes" command. |

There is also similar mechanism provided by the Globus Toolkit (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 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 irrespective of their access interfaces or mechanisms.

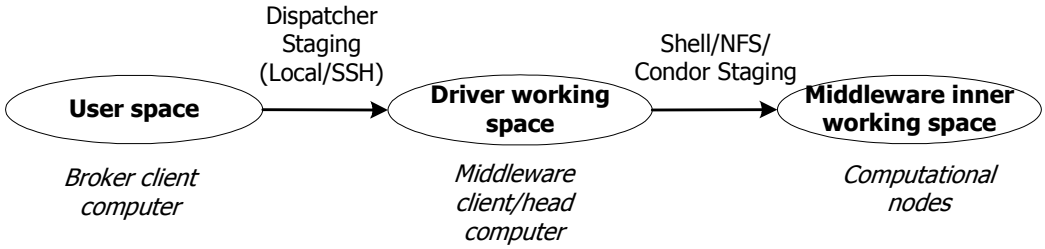


**Figure 3 - The three spaces.**

Figure 4 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 connect to middleware, in order to help stage-in/out, job execution as well as retrieving the job handle for further queries. For different middleware, the driver can adapt to their local shell command interfaces. The next sections

show how this happens for each middleware.

For the Fork driver, it stages file, between the driver working space and the middleware inner working space by the shell command "ln –s" (for making symbol links in UNIX-like systems) for stage-in and "mv" 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's 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.

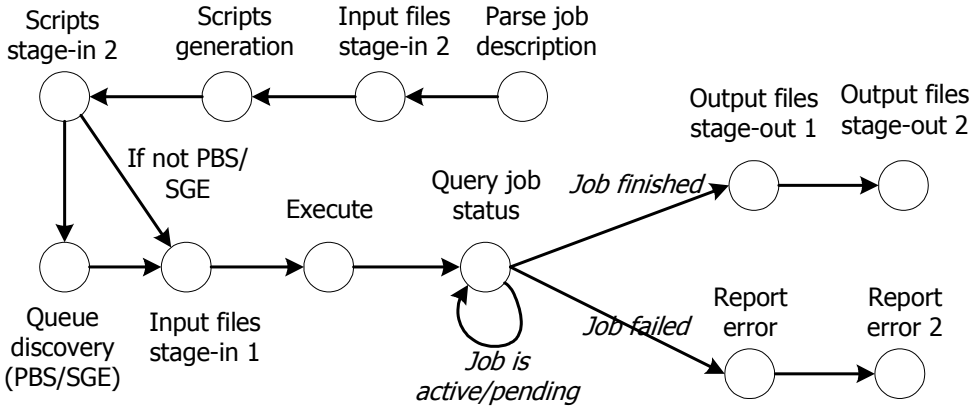


**Figure 4 - State machine of the driver flow. For Stage-In and Stage-Out: 1 represents between the user space and driver working space. 2 represents between the driver working space and middleware inner working space.**

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. But the execution scripts (needed by PBS and SGE, describing the job configuration) need to be staged to the correct directory specified by the middleware and all file attributes need to be updated so that the files could be accessible under the middleware execution environments. Also, 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, user can either specify the queue(s) to be used by the job submission, or let the broker to 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 running "qsub" (the job submission interface in both PBS and SGE), to submit the execution script into 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 the running the command "qstat" (the job status query interface in both PBS and SGE) through a shell script and analyzing the stdout stream.

For Condor driver, the staging of both the input and output files is handled automatically by Condor. What the driver needs to do is simply specify all the files in the Condor execution script (needed by Condor for describing the job configuration). Then the driver generates a job submission shell script to submit the execution script to Condor and retrieves the Condor PID, which is regarded as the job handle for Condor jobs. After the job submission, the broker queries the job status using a shell script running "condor_q" (the job status query interface in Condor). Condor provides a selection for universes. The standard 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 obviously unpractical for most jobs without the source code. So the broker will simply submits the job into a universe called the "vanilla universe".

In the next section, we provide design and implementation details of Gridbus-Xgrid adaptor only even so similar approach has been used in the implementation of adaptors for Condor, PBS, and SGE.

## 4. Design and Implementation for Xgrid

Nowadays, grid technologies are available for UNIX based OSes and Windows, while Xgrid is an intent to leverage 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 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. We describe in this section how the interface with Xgrid is implemented and further we discuss how it can be done for diverse middleware.

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`. The implementation of the `JobWrapper` class has to provide the methods necessary to submit a given job to the corresponding middleware, while the implementation of the `ComputeServer` represents a compute resource and offers means to query the job's execution status, as described in Figure 5.

Gridbus Broker aims at being platform independent and client-centric. Following this approach, we assumed that the broker can be installed on computer running 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 other UNIX class operating systems. Therefore, we decided to proceed according to the following steps:

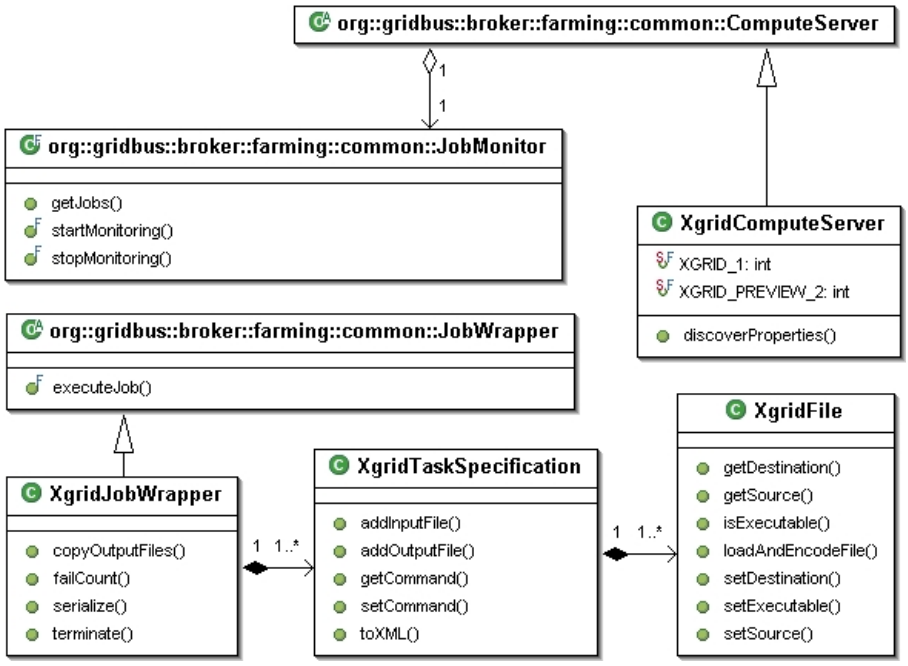


**Figure 5 - Class diagram of the Xgrid adaptor.**

- The user performs the job submission using Gridbus Broker. At this time the user can inform the script or command to be executed on the remote node as well as the necessary input files. Gridbus broker uses its own XML-based parametric language for job specification, which the user can utilize.
- The submission is carried out using the command line application on an Xgrid client. This command line application provides means to authenticate with a controller, to submit jobs, to stop execution and verify the status of ongoing jobs. 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. When the job execution is finished, the results will be available in the local working directory. The broker node can use this node directly if it is being run natively on the same node, or via SSH [15] (Figure 6), when it is running on remote and possibly under a different operating system.

- Since the job completes in Xgrid, the results are copied back from the xgrid client node to the broker node.

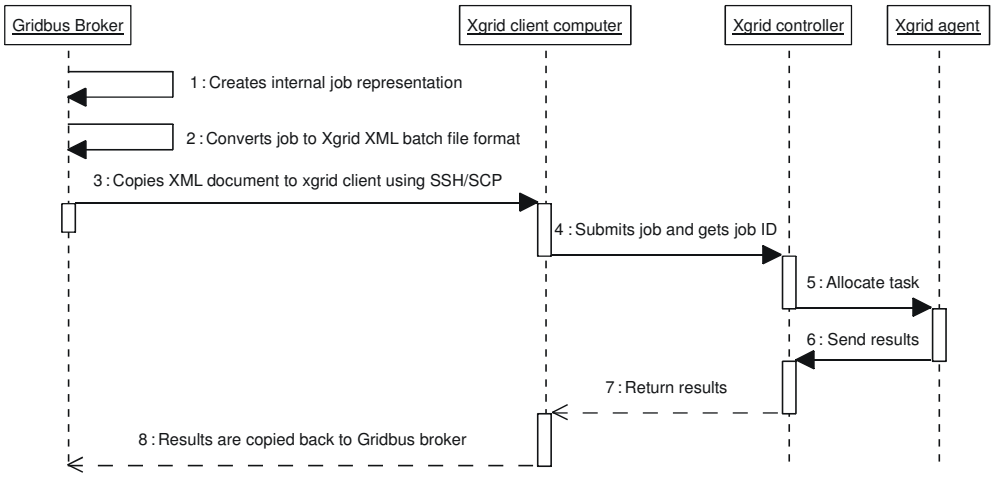


**Figure 6 - Abstract view of the Xgrid interface for Gridbus Broker.**

To clarify this process we provide a description about how the broker interacts with Xgrid.

### 4.1. XgridJobWrapper

The broker uses its own representation for jobs. The XgridJobWrapper class is used to convert the job description from the broker representation to the Xgrid's format and submit it. In the broker, a job can consist of tasks such as SUBSTITUTE, COPY and EXECUTE. The copy tasks represent input files which are required by the job as well as files that have to be copied back since the job has been completed. For Xgrid, if the broker is running on a remote computer, the files will be copied back via by SSH/SCP. Otherwise, the files are saved into the broker's temporary directory.

The Xgrid Version 1.0 accepts batch files which are basically XML documents containing serialized input files, description of commands to be executed, arguments, etc. 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 in the version 1.0 case, and as separate files in the Technical Preview 2 version. The two methods are described as follows:

Method 1:

```
if(server.getVersion() == XgridComputeServer.XGRID_1){
  String <XMLSource> = ...
  submitCommand = "xgrid -job batch " + job.getRemoteDir() + xmlName;
  ...
  // Serialize all input files and the executable into a XML document
  serialize(<XMLFile>);
  ...
  if(server.useSSH())
   server.getSSHSession().scpTo(<XMLSource>, <XMLDestination>);
  }
}
```

Method 2:

```
else if (server.getVersion() == XgridComputeServer.XGRID_PREVIEW_2){
  submitCommand = "xgrid ...";
  ...
  while(<There are files to be copied>){
   ...
   if(server.useSSH()){
    session.scpTo(<FILESource>, <FILEDestination>);
   }
   else{
    // Copy files to the broker local directory
   }
```

```
  }
}
```

Table 2 presents the example of a job description for Xgrid resulting from the conversion from the broker's job format to the one used by the version 1.0 of Xgrid.

Since the input files have been staged to the Xgrid client node, the job can be carried out. It implies in invoking the xgrid command line passing the XML document as an argument or the corresponding arguments in the previous version of Xgrid. When the job is submitted, a job ID is obtained from Xgrid which will be used to monitor the job execution.

```
InputStream input;
String submitCommand = <Different for each version of Xgrid>;
if(server.useSSH()){
  input = session.executeAndGetResult(submitCommand);
}
else{
  input = Runtime.getRuntime().exec(submitCommand).getInputStream();
}
BufferedReader reader = new BufferedReader(new InputStreamReader(input));
...
xgridJobID = Integer.parseInt(<String containing the ID provided by Xgrid>);
```

After the job submission is successful, the job monitor for this job is started and the job status is set to *submitted*. These details are described in the broker manual available in [18].

**Table 2 - Xgrid XML Job definition.**

**Xgrid XML Job Representation**

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Com-
puter//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-
1.0.dtd">

<plist version="1.0">
 <array>
   <dict>
     <key>name</key>
     <string>j1</string>
     <key>inputFiles</key>
     <dict>
       <key>GB2.C36A78DA-1C11-9485-CBCF-
07D2B009C953/DB1115811977171.j1/calc</key>
       <dict>
         <key>fileData</key>

<string>f0VMRgEBAQAAAAAAAAAAAAIAAwABAAAAAMIY
...==</string>
         <key>isExecutable</key>
         <string>YES</string>
       </dict>
```

```
     </dict>
     <key>taskSpecifications</key>
     <dict>
       <key>0</key>
       <dict>
         <key>command</key>
         <string>GB2.C36A78DA-1C11-9485-CBCF-
07D2B009C953/DB1115811977171.j1/calc</string>
         <key>environment</key>
         <dict>

<key>XGRID_CONTROLLER_HOSTNAME</key>
           <string>xgrid1.cs.mu.oz.au</string>

<key>XGRID_CONTROLLER_PASSWORD</key>
           <string>controllerpass</string>
         </dict>
         <key>arguments</key>
         <array>
           <string>1</string>
           <string>0</string>
         </array>
         <key>inputFileMap</key>
         <dict>
           <key>textfile</key>
           <string>GB2.C36A78DA-1C11-9485-
CBCF-
07D2B009C953/DB1115811977171.j1/calc</string>
         </dict>
       </dict>
     </dict>
   </dict>
 </array>
</plist>
```

### 4.2. XgridComputeServer

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 regarding the resource and the job status by using the tools specifics to the corresponding middleware. For Xgrid, a ComputeServer represents a cluster of Mac computers. A brief description about what happens when the job status is queried is presented below.

```
int status = Job.UNKNOWN;
XgridJobWrapper wrapper = (XgridJobWrapper)j.getJobHandle();
if(wrapper.getXgridJobID() == -1)
  status = Job.FAILED;
else{
  ...
  // execute xgrid querying the job status and parses the
  ...
  if(<Xgrid_status>.indexOf("running")!=-1)
     status = Job.ACTIVE;
  else if(<Xgrid_status>.indexOf("prepared")!=-1)
     status = Job.ACTIVE;
  else if(<Xgrid_status>.indexOf("failed")!=-1)
     status = Job.FAILED;
  else if(<Xgrid_status>.indexOf("finished")!=-1){
     if(<all output files where successfuly copied back>)
        status = Job.DONE;
     else
        status = Job.ACTIVE;
  }
  else
     status = Job.FAILED;
}
```

## 5. Performance Evaluation

To evaluate the interfaces to Xgrid, we ran some experiments. In this section, we describe the experiments 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 based located in Howard Florey Institute for Neuroscience at the University of Melbourne. This cluster has currently 13 nodes which are connected by a Fast Ethernet LAN. The capabilities of the controller and the nodes are described in Table 3. 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 also used for other purposes such as mail and web servers, we are unable to install Xgrid version 1.0 in this cluster at the moment. For future experiments, we aim at using version 1.0, which is a production stable release.

**Table 3 - List of resources in the Xgrid testbed.**

| | |
|---|---|
| Number of nodes/computers | 13 |
| Number of compute nodes | 12 |
| Number of control node(s) | 1 |
| Number of processors per node | 2 |
| Processor architecture | Power PC |
| Processor frequency | 1.8Ghz |
| RAM per node | 2.5GB |
| Local disk space per node | 150 GB |
| Operating System | Mac OS 10.3.9 |

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 which is copied to the Xgrid client node. This application generates 16,000 digits for PI and takes about 7 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 which are copied back consist in this output file, the standard output that is of size 0 and stderr that will be of size 0 in case of success.

The scenario evaluated with Xgrid was the execution of a 100 jobs created as a result of formulation of application as parameter-sweep application using XPML (XML-based parametric modeling language) supported by Gridbus broker. During the execution we measured how many jobs were completed over the execution time. The result of this evaluation is presented in Figure 7. 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. The job that has failed is due to have failed twice during the job submission or when the broker queries the job status.

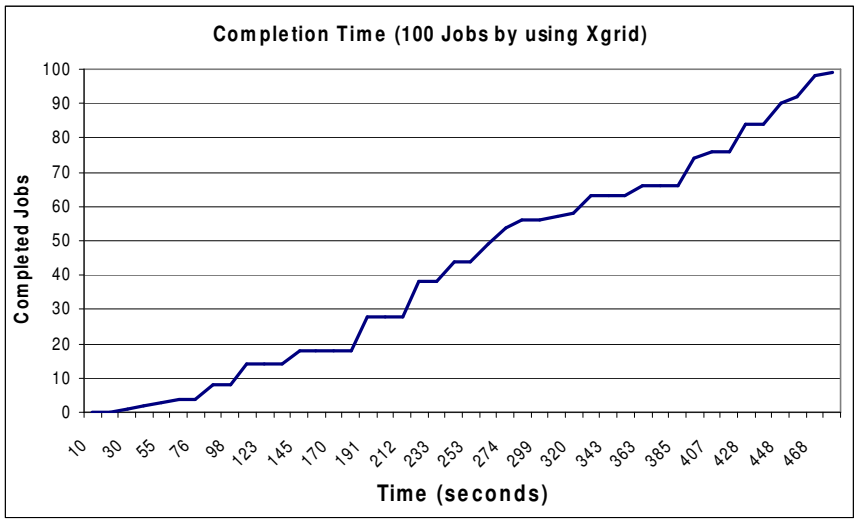


**Figure 7 - Execution time (100 jobs by using Xgrid TP2).**

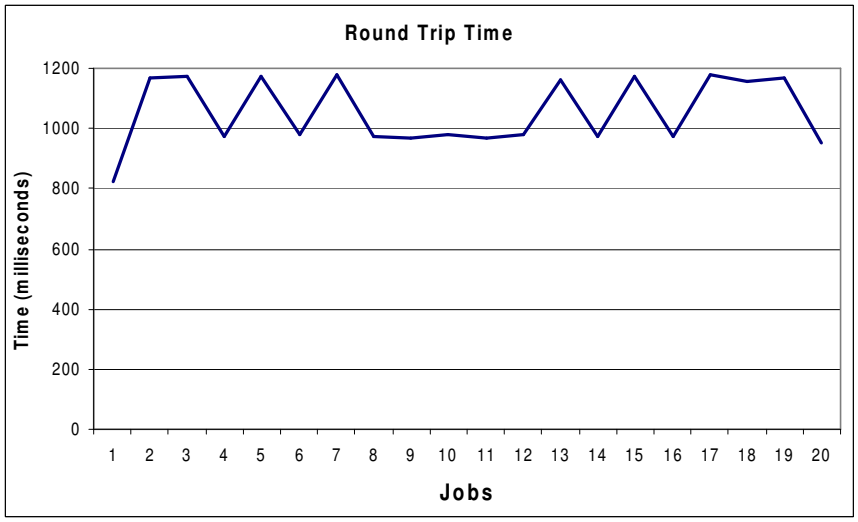


**Figure 8 - Round trip time by using SSH.**

The interfaces for the middleware described in this paper should use SSH for job submission. The SSH/SCP is used when the broker is installed on a remote computer and there is no library for Java to interface with this middleware. The Figure 8 shows the round trip time in milliseconds for Xgrid when submitting jobs to an Xgrid client by using SSH from a computer in the same local area network. We have evaluated just the time from the submission

triggered by the broker node until the response from the computer acting as an Xgrid client. The round trip time can present some variation, since the network has been utilized by its users for other purposes. According to these results, when using SSH all the jobs will have increase in the execution time in about 1 second in the testbed's local area network.

### 5.1. The PBS/SGE Testbed and Results

We have used two clusters in this evaluation. The first one, the same as the above, composed of Mac OS X based located in Howard Florey Institute for Neuroscience at the University of Melbourne, but we used SGE to manage its resources. The second cluster, called Manjra, is composed of 12 Linux nodes with Intel Pentium 4 CPUs, 2.40GHz and with 512MB of memory and it is located in another building (GRIDS Lab) at the University of Melbourne. We used PBS to manage Manjra cluster resources. The job submission to PBS was done by using SSH/SCP and the results are presented in Figure 9. As the second cluster is under 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 former described cluster, this application takes around 3-5 minutes to finish its execution by using Java 1.4.

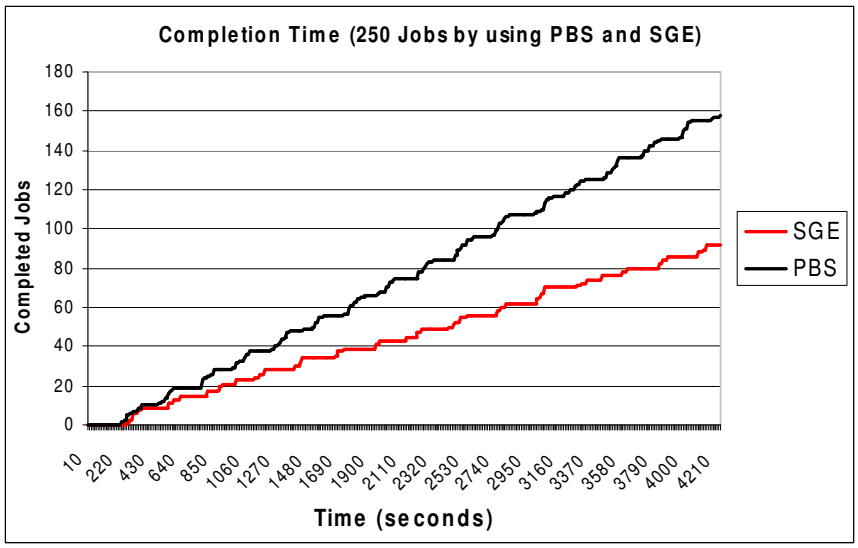


**Figure 9 - Execution time (250 jobs by using SGE and PBS).**

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 and although this release of Xgrid present some bugs, we are satisfied with the results. However, we hope that if the broker did not have to submit the jobs that fail during the first try when submitting to Xgrid, we could have better results. We hope to carry out experiments by utilizing the version 1.0 of Xgrid as soon as we update our grid.

## 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 pools of Condors by setting up the manager of a cluster to accept requests from another manager. Such approach is commonly called flock of Condors [23]. In this case, a pool *B* will send jobs to a pool *A* if the local resources are unavailable or in use.

TrellisWeb [24] provides an interface to a placeholder [25] scheduling that address resource scheduling and allocation, single log on and access control. The system is integrated with PBS, SGE and and IBM's LoadLeveler and leverages this tools to 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) [26] 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 API, GAT aims at enabling the easy development of "grid-aware" applications. This API is targeted to the development of portals and so can enable a range of grid applications. Although this project share many characteristics with our approach, it is not target at integrating the API with enterprise middleware.

Our integration still differ from the works mentioned above in that Gridbus broker does not provide the integration with only one toolkit, unlike the flock of Condors, but several enterprise and global grid middleware. Furthermore, Gridbus broker provides a uniform interface which 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

The Gridbus project has 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 APIs for users to develop their applications for enterprise and global grids.

Several interfaces for different middleware have been developed. However, we presented in detail the one for Xgrid. By integrating such technologies with the broker, we are aiming at leveraging global grids. Users can use their Macintoshes as well as other computers by the integration with other systems.

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

As future work we propose to adapt the interface to Xgrid to use BEEP protocol to communicate directly with the Xgrid controller, avoiding using the 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. We also intend to demonstrate the broker working with several middleware at the same time.

## References


[1] I. Foster and C. Kesselman, Eds., The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers, 1999.

[2] Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton, A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, Proceedings of the 2nd International Workshop on Middleware for Grid Computing (Co-located with Middleware 2004, Toronto, Ontario - Canada, October 18, 2004), ACM Press, 2004, USA.

[3] J. Almond, D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce. Future Generation Computer Systems 613, Pages 1-10, 1999.

[4] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, Alchemi: A .NET-Based Enterprise Grid Computing System, Proceedings of the 6th International Conference on Internet Computing (ICOMP'05), Las Vegas, USA, June 27-30, 2005.

[5] D. Kramer and M. Maclnnis, Utilization of a local grid of Mac OS X-based computers using Xgrid, In Proceedings of the 13th IEEE International Symposium on High performance Distributed Computing, Pages 264-265, June 2004.

[6] P. Asadzadeh, R. Buyya, C. Ling Kei, D. Nayar, and S. Venugopal, Global Grids and Software Toolkits: A Study of Four Grid Middleware Technologies, High Performance Computing: Paradigm and Infrastructure, Laurence Yang and Minyi Guo (editors), ISBN: 0-471-65471-X, Wiley Press, New Jersey, USA, June 2005.

[7] R. Buyya and S. Venugopal, The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report, Proceedings of the First IEEE International Workshop on Grid Economics and Business Models (GECON 2004, Seoul, Korea), Pages 19-36, ISBN 0-7803-8525-X, IEEE Press, New Jersey, USA, April 23, 2004.

[8] M. Rose, The Blocks Extensible Exchange Protocol Core, RFC 3080, March 2001.

[9] M. Rose, Mapping the BEEP Core onto TCP, RFC 3081, March 2001.

[10] B. Warrene, Stanford University Lab Builds an Xgrid. Macnews: Hardware. August 2004. http://www.macnewsworld.com/story/35786.html.

[11] S. Bowness, Xgrid and Cross Platform Grid Computing, 440 Project Report, University College of the Fraser Valley, April 2005.

[12] Apple Developer Connection, Xgrid: High Performance Computing for the Rest of Us, Updated: March 2005. http://developer.apple.com/hardware/hpc/xgrid_intro.html

[13] Mac OS X Server - Xgrid Administration, 2005. http://images.apple.com/server/pdfs/Xgrid_Admin_v10.4.pdf.

[14] A. Abbas, Grid Computing: A Practical Guide to Technology and Applications, Charles River Media, 408 Pages. ISBN:1584502762, 2004.

[15] JSch - Java Secure Channel. http://www.jcraft.com/jsch/

[16] G. Fox, D. Walker. e-Science Gap Analysis. June, 2003. http://www.grid2002.org/ukescience/gapresources/GapAnalysis30June03.pdf

[17] Enterprise Grid Alliance Reference Model v1.0, April 2005. http://www.gridalliance.org

[18] K. Nadiminti, S. Venugopal, H. Gibbins, and R. Buyya, The Gridbus Grid Service Broker and Scheduler (2.0) User Guide, Technical Report, GRIDS-TR-2005-4, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, April 22, 2005.

[19] I. Baird, Grids in Practice: a Platform perspective, MIDDLEWARESpectra, Pages 25-33, June 2003. http://www.middlewarespectra.com/grid/grid_contents.htm.

[20] The Condor Project Homepage. http://www.cs.wisc.edu/condor/

[21] Veridian Systems. OpenPBS: The portable batch system software. Veridian Systems, Inc., Mountain View, CA. http://www.openpbs.org/scheduler.html.

[22] Sun Grid Engine (SGE): A cluster resource manager. http://gridengine.sunsource.net/.

[23] A.R. Butt, R. Zhang, Y. Charlie Hu, A Self-Organizing Flock of Condors, Proceedings of the 2003 ACM/IEEE conference on Supercomputing, Pages 42, IEEE Computer Society, Washington, DC, USA, 2003.

[24] C. Pinchak, P. Lu, J. Schaeffer, M. Goldenberg, The Canadian Internetworked Scientific Supercomputer, 17th International Symposium on High Performance Computing Systems and Applications (HPCS), Pages 193-199, Sherbrooke, Quebec, Canada, May, 2003.

[25] C. Pinchak, P. Lu, J. Schaeffer, M. Goldenberg, Practical Heterogeneous Placeholder Scheduling in Overlay Metacomputers: Early Experiences. In Proc. 8th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), Pages 85–105, Edinburgh, Scotland, UK, July 24, 2002.

[26] 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, B. Ullmer, The grid application toolkit: toward generic and easy application programming interfaces for the grid, Proceedings of the IEEE, Volume 93, Issue 3, Pages 534-550, March 2005.

[27] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2), Pages: 115-128, 1997.