The Design and Implementation of a Dynamic Resource Broker Framework in a Grid Environment

Kun-Ming Yu¹, Li-An Yang², Zhi-Jie Luo³, Jiayi Zhou⁴
Department of Computer Sciences and Information Engineering, Chung Hua University
Hsin-chu, Taiwan 300, ROC.
Email: yu¹@chu.edu.tw, {laving², kevin³, jyzhou⁴}@pdlab.csie.chu.edu.tw

Abstract

Owing to the recent great advances in Grid Computing, computational Grids are emerging for solving grand challenge applications. Grid development involves the efficient management of heterogeneous, geographically distributed, and dynamically available resources. However, the huge and complex architecture of grid software is still immature and difficult to user, especially in a large grid computing environment. In the paper, we proposed and developed a resource broker framework to help a grid user which can easily access to the suitable grid resources and submit jobs without knowing any resource information in the system. The proposed resource broker runs on the top of the Globus Toolkit. It can dynamically provide the corresponding information for the grid users in advance. The experimental results show that the proposed resource broker has superior performance compared with random as well as round-robin method.

Keyword: resource broker, grid, globus toolkit

1 Introduction

Grid computing can mean different things to different individuals. That can divide into several kinds according to different application. Among them, the more common one is data grid [11] and computing grid. In the computing grid. That grand vision is often presented as an analogy to power grids where users can get access to electricity through wall sockets with no care or consideration for where or how the electricity is actually generated. In this view of grid computing [4], computing becomes pervasive and individual users gain access to computing resources (processors, storage, data, and so on) as needed with little or no knowledge of where those resources are located or what the underlying technologies, hardware, operating system, and so on.

Therefore, the key values of grid computing are in the underlying distributed computing infrastructure technologies that are evolving in support of cross-organizational application and resource sharing virtualization across technologies, platforms, and organizations. This kind of virtualization is only achievable through the use of open standards. Open standards help ensure that applications can transparently take advantage of whatever appropriate resources can be made available to them.

We focus our attention on distributed computing solutions. The goal is to create the illusion of a simple yet large and powerful virtual computer out of a collection of connected systems sharing various combinations of resources.

The open source Globus Toolkit [10] is a fundamental framework enabling technology for the "Grid", letting people share computing power, databases, and other resources securely online across corporate, institutional, and geographic boundaries without sacrificing local autonomy [1, 3, 5]. The toolkit includes software for security, information infrastructure, resource management, data management, communication, fault detection, and portability. It is packaged as a set of components that can be used either independently or together to develop applications. Each organization has unique modes of operation, and collaboration between multiple organizations is hindered by incompatibility of resources such as data archives, computers, and networks.

In Globus Toolkit environment, users usually perform some queries to check how busy of the grid system is, to see how the submitted jobs are progressing [4, 6, 7], and to look for resources on the grid. Globus Toolkit usually provides command-line tools for queries. Command-line tools are especially useful when the user wants to write a script that automates a sequence of actions. For example, the user might write a script to look for an available resource,
submits a job to it, watches the progress of the job, and presents the results when the job has finished.

Job submission usually consists of three parts. First, some input data and possibly the executable program or execution script file are sent to the machine to execute the job. Alternatively, the data and program files may be pre-installed on the grid machines or accessible via a network file system. When the grid consists of heterogeneous machines, there may be multiple executable program files, each compiled for the different machine platforms on the grid. A nice feature provided by some grid systems is to register these multiple versions of the program so that the grid system can automatically choose a correctly matching version to the grid machine that will run the program.

Second, the job is executed on the grid machine. The grid software running on the donating machine executes the program in a process on the user’s behalf. It may use a common user ID on the machine or it may use the user’s own user ID, depending on which grid technology is used. Rights to access files and other resources on the grid machine may be restricted.

Third, the results of the job are sent back to the submitter. In some grid technologies that do not automatically stage the output data back to the user, the results must be explicitly sent to the user, perhaps using network file system.

But it spends much time to query grid system and finds out free resources for submitting job, and it’s too difficult to find the best computing node from those free resources. To overcome the drawbacks, we proposed a resource broker framework to solve the problem. In the proposed framework, when user submits a job the system will search the most suitable computing node and submits the job automatically. The experimental results show our proposed architecture can effectively improve system efficiency and reduce job execution time.

The rest of the paper is organized as follows. Section 2 will introduce the Globus Toolkit and its related components. The proposed resource broker framework will be described in detail in Section 3. Section 4 will give the experimental results. The conclusions will be given in Section 5.

2 Related Work

2.1 OGSA

The Global Grid Forum has published the Open Grid Service Architecture (OGSA) [9]. To address the requirements of grid computing in an open and standard way requires a framework for distributed systems that support integration, virtualization, and management. Such a framework requires a core set of interfaces, expected behaviors, resource models, and bindings.

OGSA defines requirements for these core capabilities and thus provides general reference architecture for grid computing environments [2]. It identifies the components and functions that are useful if not required for a grid environment. Though it does not go to the level of detail such as defining programmatic interfaces or other aspects that would guarantee interoperability between implementations, it can be used to identify the functions that should be included based on the requirements of the specific target environment.

2.2 OGSI

As grid computing has evolved it has become clear that a service-oriented architecture could provide many benefits in the implementation of a grid infrastructure. The Global Grid Forum extended the concepts defined in OGSA to define specific interfaces to various services that would implement the functions defined by OGSA.

More specifically, the Open Grid Services Interface (OGSI) defines mechanisms for creating, managing, and exchanging information among Grid services. A Grid service is a Web service that conforms to a set of interfaces and behaviors that define how a client interacts with a Grid service.

These interfaces and behaviors, along with other OGSI mechanisms associated with Grid service creation and discovery, provide the basis for a robust grid environment. OGSI provides the Web Service Definition Language (WSDL) definitions for these key interfaces.

2.3 Globus Toolkit

The Globus Toolkit is the defector standard for grid computing [10]. The Toolkit is deployed in large testbed and used to complement a variety of applications. It provides a simple, well-defined interface to a wide range of services supported on a highly heterogeneous mix of systems [8].

The Globus Toolkit is a set of components from which a developer can select to meet their need. The components include basic services for security, information, resource management, storage and communication, each service is distinct and has well-defined interfaces so they can be incorporated into applications or tools in an incremental fashion.

The layered architecture of the Toolkit is analogous to an hourglass. At the neck of the hourglass
is a small set of core abstractions and protocols from which many different high-level behaviors can be mapped onto many different underlying technologies. These abstractions and protocols provide uniform access to diverse implementations of local services, and building blocks upon which global services can be built. A local site need only provide these local services. Global services can be built without knowledge of local implementation.

Globus Toolkit 4 is a collection of open-source components. Many of these are based on existing standards, it provides five category components: Common runtime components, Security, Data management, Information services, and Execution management. Each component describe as following:

- **Common runtime components**: that consist of libraries and tools needed by both types of implementations and used by most of the other components.

- **Data management**: Includes GridFTP and Reliable File Transfer (RFT) components, the GridFTP facility provides secure and reliable data transfer between grid hosts. And the RFT provides a Web service interface for transfer and deletion of files.

- **Information services**: Include Monitoring and Discovery Services (MDS) component, the MDS are mainly concerned with the collection, distribution, indexing, archival, and it provides grid information such as the resources that are available and the state of the computational grid.

- **Execution management**: Include Grid Resource Allocation and Management (GRAM) component, the GRAM provides a standard interface for computational grid tools and applications to express resource allocation and management re-quests.

- **Security**: All components communication based on the security component.

When users want to submit job to the grid system, in traditional, user must precede many complicate procedures. Therefore, we proposed a Resource Broker to resolve this problem, users can utilize our proposed Resource Broker to submit job and can automatically select the most appropriate resource to execute the submitted jobs.

3 Implementation

In this section, we will describe Resource Broker Module and Resource Manager Selection Module in detail. Such as above-mentioned description, when users want to submit jobs by Globus Toolkit, the user usually needs many complicate procedures, and difficult to select an appropriate computing resource. Therefore, we proposed a Resource Broker not only to automatically complete job submission procedures, but also to select the most appropriate computing resource to improve grid system performance.

3.1 Resource Broker

The resource broker is a middleware that supplies distributed clients with job execution at the more suitable computing node in a heterogeneous computing environment. We combine many GT components to construct resource broker. Moreover, we provide a friendly command-line user interface for job submission. Through this combination can reduce the user complex process of submit job, select appropriate resource to execute the job, and select appropriate resource that can efficiently reduce the response time, turnaround time and increase the throughput. Resource discovery is the first task of resource broker. The main goal is to identify a list of authorized hosts that are available to users. Most resource discovery algorithms interact with some kind of grid information service (GIS), like MDS (Monitoring and Discovery Service) in Globus. Once the list of possible target hosts is known, the second job of broker is selecting those resources that are expected to meet the time or cost constraints imposed by the user. In order to fulfill the user time restrictions the resource broker has to gather system's information dynamically about resource accessibility, system workload, and network performance, etc.

The next stage of resource brokering is resource selection. Most of the grid systems use performance-guided selection, since they try to find a job minimizes the execution time.

![Figure 1 The control flow of resource broker](image)

Figure 1 shows the control flow of resource broker as follows:

1. User prepares their executable jobs and input
1. Data. Then submit a job to resource broker.
2. The resource broker got request and then send information query to Execution Host.
3. The Execution Host returns information such as CPU speeds, CPU loading, Memory size to resource broker.
4. After got information, the resource broker sends a request to resource manager.
5. The Resource Manager analyzes and selects the suitable nodes and then replies.
6. The broker submits job to the selected nodes.
7. The resource broker creates a connection to the server node, through the GRAM service, job manager run the file for fork process on the selected nodes, and then run.
8. When process complete execution, the computing result will be sent back to the user.

Figure 2 shows the framework of the proposed resource broker, it is build on top of Globus Toolkit. It makes uses of Globus services, such as resource allocation, information.

The resource broker framework runs at the top of Globus compatible components, which simplifies the user interface with the Grid, and provides the mechanisms for efficient execution of jobs on the Grid with dynamic adaptation. From the user point of view, the Resource Broker framework consists of two main components: Command-Line User Interface and Resource Manager Selection Module. The Command-Line User Interface provides user friendly job submission. And the Resource Manager Selection Module provides dynamic computing node selection. Because every component is forms of the module, so user can convenient to construct their experiment environment, such as use round-robin, random or other selection methods for Resource Manager Selection Module.

3.2 Resource Manager Selection Module

In order to adapt the execution of a job to its dynamic demands, the application must specify its host requirements through a requirement expression. The application could define an initial set of requirements and dynamically change them when more, or even less, resources are required. Also, in order to prioritize the resource that fulfills the requirements according to its runtime needs, the application must specify its host’s preferences through a ranking expression. A compute-intensive application would assign a higher rank to those hosts with faster CPUs and lower load, while a data-intensive application could benefit those hosts closer to the input data.

The requirement expression and ranking expression files are used by the resource selector to build a list of potential execution hosts. Figure 3 shows the resource selection process. Initially, available computing resources are discovered by accessing the GIIS server and those resources that do not meet the user-provided requirements are filtered out. At this step, an authorization test is performed to guarantee user access. Then, the resource is monitored by accessing its local GRIS server. The information gathered is used to assign a rank to each candidate resource based on the user-provided preferences. Finally, the resultant prioritized list of candidate resources is used to dispatch the jobs.

In order to reduce the information retrieval overhead, the GIIS and GRIS information is locally cached at the client host and updated independently in order to separately determine how often the testbed is searched for new resources and the frequency of resource monitoring. In the following experiments we set the GIIS cache timeout to 5 minutes and the GRIS cache timeout to 1 minute.

Figure 3 shows the resource selection process.
4 Experimental Results

4.1 Experimental Environment

We construct a grid computing testbed which includes four Linux PCs.

Server A: With Intel P4 2.8G, 768MB RAM and Linux kernel is 2.6.8-3, resource broker is installed on this server.

Server B: With dual Intel P4/Xeon HT 3.2G, 1G RAM and Linux kernel is 2.6.8-3

Server C: With Intel P4 3.2G, 512MB RAM and Linux kernel is 2.4.27

Server D: With dual AMD 1.2G, 1G RAM and Linux kernel is 2.6.8-3

Each site is running on Debian Linux and installed Globus Toolkit v4.0.2. Figure 4 shows the experimental environment. In order to verify the performance of our proposed rank index method, we implemented our rank index method along with Random and Round Robin submission methods. The Random method means when job submit, resource management choose a node randomly from the server list, and the round-robin select node according to round-robin policy.

4.2 Fibonacci benchmark

In the experiment, the jobs are run on our grid testbed. We will run programs with CPU bounded type to observe the performance. Also we execute the same job on single machine environment to compare the execution time. Table 1 and Figure 5 show the experimental results.

![Figure 4 The experimental environment](image)

4.3 A Pi application benchmark

The second benchmark program we adopted is Pi application. It computes the value of by numerical integration.

\[
\int_0^1 \frac{1}{1 + x^2} dx = \tan^{-1}(1) = \frac{\pi}{4}
\]

We can compute by integration the function from 0 to 1. We compute an approximation by dividing the interval [0, 1] into some number of subintervals and then computing the total area of these rectangles by having each process compute the areas of some subset. The experimental benchmark used in this work is based on a Pi application.

Table 2 and Figure 6 compare the three different selection methods. As the result, Pi application is a CPU intensive application. The results show that the resource broker can achieve better performance under this strategy. To compare to the random or round-robin, it can reduce 50% computing time at most compared to random or round-robin method. If the random or round-robin nodes with poor CPU speed, it will result in poor computer performance. When the number of tasks is 12, the execution time is increased. Also, when the number of task is greater than the number of processor, the resource broker need wait more time to transfer file and to complete the job’s execution.

![Figure 5 Fibonacci turnaround time](image)

**Table 1** Fibonacci turnaround time

<table>
<thead>
<tr>
<th>Task number</th>
<th>Random</th>
<th>Round-robin</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>638</td>
<td>611</td>
<td>600</td>
</tr>
<tr>
<td>2</td>
<td>724</td>
<td>459</td>
<td>445</td>
</tr>
<tr>
<td>4</td>
<td>371</td>
<td>412</td>
<td>223</td>
</tr>
<tr>
<td>8</td>
<td>275</td>
<td>262</td>
<td>193</td>
</tr>
<tr>
<td>12</td>
<td>310</td>
<td>332</td>
<td>201</td>
</tr>
</tbody>
</table>

**Table 2** Pi application benchmark

<table>
<thead>
<tr>
<th>Job number</th>
<th>Single machine</th>
<th>Rank index</th>
<th>Round robin</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>148</td>
<td>89</td>
<td>109</td>
</tr>
<tr>
<td>100</td>
<td>296</td>
<td>121</td>
<td>172</td>
</tr>
<tr>
<td>200</td>
<td>597</td>
<td>256</td>
<td>402</td>
</tr>
<tr>
<td>300</td>
<td>894</td>
<td>322</td>
<td>599</td>
</tr>
<tr>
<td>400</td>
<td>1188</td>
<td>426</td>
<td>787</td>
</tr>
<tr>
<td>500</td>
<td>1485</td>
<td>565</td>
<td>1008</td>
</tr>
</tbody>
</table>
5 Conclusions

We proposed a resource broker for a grid environment, the proposed can dynamically allocate the most suitable computing node with better resources for the user. Also, the proposed Resource Broker can effectively integrate the corresponding information and reduce unnecessary operations. The resource broker runs on top of the Globus Toolkit and has dynamic resource information monitoring capability.

In the experimental results, we can find that the proposed Resource Broker has better performance compared with the traditional method. Also, resource broker can save 50% of the computing time with other methods since it can select the most appropriate computing nodes by dynamically gather information from every computing node. It shows that our proposed Resource Broker can provide more convenient and efficient execution environment for grid system.

References