Design and Implementation of an OGSI-Compliant Grid Broker Service

Young-Seok Kim† , Jung-Lok Yu† , Jae-Gyoon Hahm‡ , Jin-Soo Kim† , and Joon-Won Lee†
Computer Science Division, Korea Advanced Institute of Science and Technology (KAIST) †
Korea Institute of Science and Technology Information (KISTI) ‡
{kimys, jlyu}@camars.kaist.ac.kr, jaehahm@kisti.re.kr, { jinsoo, joon}@cs.kaist.ac.kr

Abstract

Grid computing promises the ability to share geographically and organizationally distributed resources to increase effective computational power and resource utilization. However, for the grid computing to be successful, it is very important to provide middleware services that assist grid users to easily interact with grid environments.

In this paper, we have designed and implemented a new general-purpose OGSI-compliant Grid resource broker service to hide the underlying complexity of the Grid resources from Grid users and to meet not only Grid user's requirements but also resource owner's policies. It focuses on the discovering and scheduling dynamic resources scattered across multiple organizations. Furthermore, it can be integrated with various scheduling services. We also present experimental results and demonstrate the effectiveness of our Grid broker service.

1. Introduction

Grid computing [1] is an approach to distributed computing that provides unlimited high-end computing resources to Grid users without regard to their physical locations. A Grid can be defined as a collection of distributed computing resources available over local or wide area networks that appears to Grid users as one large virtual computing system. The ultimate goal of the Grid is to create dynamic virtual organizations (VOs) through secure, coordinated resource sharing among individuals, institutions, and resources [2]. Grid computing technology has been widely and successfully used to solve large-scale science and engineering problems.

Recently, Grid computing has started to leverage Web Services [3] technology to define standard interfaces for constructing Grid environments. The *Open Grid Services Architecture* (OGSA) [4,5] aims to define a new common and standard architecture for Grid-based applications. The OGSA views a Grid as an extensible set of Grid Services that may be interoperated in various ways to meet the need of VOs. Here, a Grid Service is a Web Service that conforms to a set of interfaces and behaviors. Those interfaces and behaviors define how Grid users or applications interact with the Grid service. More specifically, the OGSA provides mechanisms: 1) for creating, naming, and discovering transient Grid Service instances, 2) for managing Grid Service lifetime, and 3) for subscribing and notifying specific service data. The *Open Grid Services Infrastructure* (OGSI) [6] is a formal and technical specification of the concepts described in OGSA. *Globus Toolkit* 3.0 (GT3) [4] is a reference implementation of OGSI specification.

Although Globus Toolkit provides many useful Grid-related services including GT3 Security Services [7], GT3 Base Services [8,9,10], and GT3 Data Services [11], the discovery and selection of suitable resources for applications in Grid environment remain challenging problems. When Grid users are to use a Grid, all processes related to resource discovery, resource selection, and resource scheduling, should be handled manually. This is because no Grid *resource broker* service is available on top of Globus Toolkit.

In this paper, we design and implement a new general-purpose OGSI-compliant Grid resource broker service. The role of the resource broker service is to find the best match between the requirements of the job and the distributed computing resources on the Grid. Our broker service hides the underlying complexity of the Grid resources from Grid users by providing automatic resource discovery and scheduling. Furthermore, during the resource discovery and scheduling procedure, we consider not only job's requirements on resources, but also resource owner's usage policies. This allows resource owners to tightly control the usage of their resources. We provide XML-

0-7803-8430-X/04/$20.00 ©2004 IEEE

based extensible schemas to represent user requirements and resource owner policies by modifying Resource Specification Language (RSL) [12] and GLUE [13] schema, respectively.

The rest of this paper is organized as follows. Section 2 overviews previous research on resource brokering and scheduling. Section 3 and section 4 discuss the system design and implementation details of our OGSI-compliant Grid resource broker service, respectively. Section 5 describes experimental results and section 6 concludes the paper.

2. Related Work

Many projects, such as AppLes, Nimrod/G, Condor-G, and EZ-Grid, have been investigating resource broker services on Grid [14].

AppLes (Application Level Scheduling) [15] focuses on developing scheduling agents for individual Grid applications. AppLes agents have an application-oriented scheduling mechanism, and use static or dynamic application and resource information to select a set of resources. However, they perform resource discovering and scheduling without considering resource owner policies. Also they do not support system-oriented or extensible scheduling policies [14].

Nimrod/G [16] broker allows managing and steering of parameter sweep applications on computational Grids. Currently, it adopts economic theories in Grid resource management and performs scheduling as part of a new framework called GRACE (Grid Architecture for Computational Economy), which includes global scheduler, bid-manager, directory server, and bid-server components. The brokering system has manually configured resource discovery mechanism. The scheduling policy is driven by user-defined requirements such as budget/deadline limitations.

Condor-G [17] is an extension to Condor [19] for Globus to allow users to harness multiple administrative domain resources. It creates a virtual Condor pool from Globus-enabled resources by using a mechanism called GlideIn, and assigns the pool to Condor users. Condor-G uses Condor matchmaking mechanism to match locally queued jobs with the resource advertised by daemons in the pool.

EZ-Grid [18] resource brokering system aims at promoting efficient job execution and controlled resource sharing across multiple sites. It performs automatic resource discovery, and uses resource provider policy framework to enable fine-grained authorization. EZ-Grid also provides deadline/budget-based scheduling according to user-specified time/cost constraints.

Our resource broker service differs from previous ones in the following aspects. First, it is a new general-purpose OGSI-compliant Grid resource broker service that performs resource discovering and scheduling with close interactions with GT3 Core and Base Services. To the best of our knowledge, no resource broker has yet been developed on top of Globus Toolkit 3.0. Secondly, the proposed resource broker service provides a general broker framework consisting of the resource scheduling service and the resource selection service. Since the resource scheduling service and the resource selection service themselves are implemented as Grid Services, they do not have to be on the same machine and it is possible to add a new resource scheduling service without change to the resource selection service. Finally, our resource broker service considers resource owner policies as well as user requirements on the resources. Resource owners specify the conditions of preferred jobs in much the same way as users specify the conditions of preferred resources for their jobs. When a job submission request arrives, our resource broker service performs matchmaking of those two conditions and selects appropriate resource candidates.

3. System Design

3.1. Overall Architecture

This subsection presents the overall architecture of the proposed Grid resource broker service. As shown in figure 1, the broker service consists of two Grid Services, Scheduling Service (SS) and Resource Selection Service (RSS). On the whole, the broker service receives *job information* as an input and produces the corresponding result of scheduling as an output. The job information is basically written in RSL-2, but has additional requirements to specify the job's preferences for resources.

RSS is responsible for discovering and selecting a set of resource candidates, which satisfy the job's requirement. SS picks out one or more resources

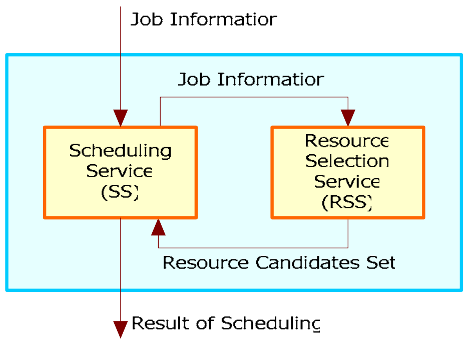

Figure 1. Overall Architecture

among resource candidates set based on its own scheduling criteria, and actually assigns the job on the target resource(s). Separating the resource selection phase (RSS) from the resource scheduling phase (SS) increases the modularity of the Grid resource broker service; it is easy to replace the default Scheduling Service and is even possible for several Scheduling Services that have different scheduling criteria to coexist in the system. More detailed description of our broker service architecture is presented in the following subsections.

3.2. Specifying Job Information

Because RSS is an OGSI-compliant Grid service, anyone, who knows the service URI and is permitted to access the service, can get a set of resource candidates from RSS. Contacting RSS requires a description of job information. Figure 2(a) shows an example of how to specify job information in XML[1].

In figure 2(a), the <input id> tag means the job's ID, which is used by SS to distinguish incoming job submission requests. Users specify the requirements on the resources for the submitting job in <resourcePreference>. The specification divides into two parts, namely <constraints> and <ranks>. The <constraints> part describes the minimal conditions that resources should meet for the job, while the <ranks> part presents the list of preferred conditions.

For example, in figure 2(a), it is specified in <constraints> that the operating system should be Linux to run the job. Note that we are using GLUE schema to describe various attributes of a resource. Not only the operating system but also available memory, CPU clock speed, hard disk capacity, and any attributes defined in GLUE schema can be used to specify resource constraints.

When multiple resources meet the <constraints>, the <ranks> part specifies the order of preference of those resources. This is very similar to Condor's matchmaker. In Condor, when multiple resources satisfy a user request, a ranking mechanism sorts available resources based on user-supplied criteria and selects the best match. However, because the matchmaker and the ClassAds language used in Condor were designed for selecting a single machine on which to run a job, it has limited applicability in the situation where a job requires multiple resources. Moreover, in contrast to Condor where the preference is represented as a single arithmetic expression, we use a more versatile point-based mechanism. In figure 2(a), if the available space in "/tmp" partition is larger than 700MB, the point is calculated by multiplying 10 by the value of available space in /tmp. All resources need not have the space larger than 700MB in /tmp, but those resources which meet the condition are more preferred by the job.

```
<?xml version="1.0"?>
<rss>
<input id="12345">
<resourcePreference>
  <constraints>
    <constraint type="glue"><element name="ce:OperatingSystem">
        <attribute name="ce:Name" value="Linux" condition="=" />
      </element>
    </constraint>
    ...                                                   Resource Constraints
  </constraints>
  <ranks>
    <rank>
      <condition type="glue">
        <element name="ce:FileSystem">
          <attribute name="ce:Name" value="/tmp" condition="=" />
          <attribute name="ce:AvailableSpace" value="700" condition=">=" />
        </element>
      </condition>
      <point>
        <operator type="multiply">
          <operand type="glue">
            <element name="ce:FileSystem">
              <attribute name="ce:Name" value="/tmp" />
              <attribute name="ce:AvailableSpace" />
            </element>
          </operand>
          <operand type="integer">
            10
          </operand>
        </operator>
      </point>                                                 Rank Points
    </rank>
    ...
  </ranks>
</resourcePreference>
<resourceCount min="1" max="10"/>
<userSN value="/O=Grid/O=Globus/OU=kaist.ac.kr/CN=Youngseok Kim"/>
</input>
</rss>
```

(a) Job information

```
<?xml version="1.0"?>
<rss>
<output id="12345">

<resourceList>
  <resource name="cc10.kaist.ac.kr" id="cc10.kaist.ac.kr">
    <rank jobPoint="95" resPoint="95"/>
  </resource>
  <resource name="cc1.kaist.ac.kr" id="cc1.kaist.ac.kr">
    <rank jobPoint="80" resPoint="90"/>
  </resource>
  <resource name="cc8.kaist.ac.kr" id="cc8.kaist.ac.kr">
    <rank jobPoint="70" resPoint="80"/>
  </resource>
  <resource name="cc7.kaist.ac.kr" id="cc7.kaist.ac.kr">
    <rank jobPoint="80" resPoint="60"/>
  </resource>
  <resource name="cc5.kaist.ac.kr" id="cc5.kaist.ac.kr">
    <rank jobPoint="50" resPoint="40"/>
  </resource>
</resourceList>
                        Resource Information used in Scheduling Service
</output>
</rss>
```

(b) Result produced by RSS

Figure 2. An Example of Job Information and the Corresponding Result from RSS

[1] In figure 2(a), we have omitted the job specification part written in RSL-2.

Another interesting elements in figure 2(a) are <resourceCount> and <userSN>. The <resourceCount> denotes the minimum and the maximum number of resources that are requested by SS. If RSS finds resources less than the minimum value, it informs SS of the failure in resource selection. Otherwise, RSS returns the list of resource candidates to SS, but the number of resources does not exceed the maximum value. Finally, the <userSN> represents the job owner's ID. It is used by RSS when a resource has different usage policies depending on the job owner.

The corresponding sample output generated by RSS is shown in figure 2(b). Note that two values, jobPoint and resPoint are associated with each resource. The jobPoint is the sum of the points that a resource has earned as a result of evaluating the <ranks> part in figure 2(a). In addition, jobPoint is normalized with respect to the maximum jobPoint of all the resources. Therefore, jobPoint indicates how much the resource is preferred by the job. On the contrary, resPoint, which is also normalized with respect to the maximum resPoint, is used to represent how much the job is preferred by the resource. How resPoint is calculated will be explained in the next subsection.

3.3. Specifying Resource Owner Policies (ROPs)

In the traditional Grid computing architecture, resources are passive and available for any jobs. The only thing resource owners can do is to permit or to restrict access rights to selected users. However, our Grid resource broker service provides a mechanism for resource owners to tightly control the usage of their resources based on the time the job is submitted, the user who submits the job, the current load level of the system, and any combinations of such policies[2].

Figure 3 shows an example of how resource owners can specify such policies. First, the line <rop id="cc10.kaist.ac.kr"> represents that this ROP (resource owner policy) is for the node, cc10.kaist.ac.kr. In the following <policy> part, we define a number of service classes based on the job submission time, the job owner, and the load level of the system. Each service class or any logical combinations of such service classes get points in the <ranks> part, according to their preferences given by the resource owner. Such points are summed up and returned to SS as a resPoint mentioned in section 3.2 (cf. figure 2(b)). When the resPoint is zero for a given job, the corresponding resource is not considered by RSS, even though the resource has the highest jobPoint among resources (i.e., it is the most preferred resource by the job).

```
<?xml version="1.0"?>
<rop id="cc10.kaist.ac.kr">
<policy>

  <group name="timeGroup" valueType="time"
      condType="range" zoneOffset="9">
    <class name="TG1">
      <range from="02:00" to="08:00"/>
    </class>
    <class name="TG2">
      <range from="any" to="any"/>
    </class>                                              Time Group
  </group>

  <group name="userGroup" valueType="sn" condType="single">
    <class name="UG1">
      <single value="/O=Grid/O=Globus/OU=kaist.ac.kr"/>
    </class>
    <class name="UG2">
      <single value="/O=Grid/O=Globus/OU=kisti.or.kr"/>
    </class>
    <class name="UG3">
      <single value="/O=Grid/O=Globus/OU=hufs.ac.kr/CN=rhee yunseok"/>
    </class>
  </group>                                                User Group

  <group name="loadGroup" valueType="load" condType="range">
    <class name="LG1">
      <range from="0" to="5"/>
    </class>
  </group>                                                Load Group

</policy>
<ranks>

  <rank>
  <!-- condition : TG1&&(UG1||UG3) -->
    <condition type="operator">
      <operator type="and">
        <operand type="string">TG1</operand>
        <operand type="operator">
          <operator type="or">
            <operand type="string">UG1</operand>
            <operand type="string">UG3</operand>
          </operator>
        </operand>
      </operator>
    </condition>
    <point type="value">100</point>
  </rank>                                     Rank Point Example 1

  <rank>
  <!-- condition : !UG1&&LG1 -->
    <condition type="operator">
      <operator type="and">
        <operand type="string">LG1</operand>
        <operand type="operator">
          <operator type="not">
            <operand type="string">UG1</operand>
          </operator>
        </operand>
      </operator>
    </condition>
    <point type="value">50</point>
  </rank>
                                              Rank Point Example 2
  <default>
    <point type="value">0</point>
  </default>
</ranks>
</rop>
```

Figure 3. An Example of Resource Owner Policy

[2] Note that our resource broker service framework can be extended easily to include other resource owner policies, if any.

In figure 3, the highest priority (100 resource points) is given to the user who belongs to kaist.ac.kr (UG1) or whose name is “rhee yunseok” in hufs.ac.kr (UG3) during office hours (TG1). Other users (~UG1) are allowed to use the resource with the lower priority (50 resource points) as long as the load average is less than 5 (LG1).

Resource owners should publish their ROPs in advance and a good place to do so is GT3 Index Service. We install a host script provider for each resource which reports its ROP to Index Service, so that the resource broker service makes use of them during the resource selection phase. Although it is somewhat complex and verbose to represent ROPs in XML, we believe a simple GUI program can assist the generation of such XML documents. This also applies to the generation of job information described in section 3.2.

4. Implementation

4.1. Resource Selection Service (RSS)

As illustrated in figure 4, RSS consists of five components: Job Information Parser, Query Processor, Index Service Agent (IS Agent), Cache Manager, and Resource Selector. When RSS receives job information from SS, Job Information Parser separates resource constraints from job specification. Then Query Processor converts resource constraints into XPath queries and passes them to IS Agent to search for the resources satisfying the job’s requirements in GT3 Index Service.

Cache Manager interacts with three other components: a local cache, Index Service, and Resource Information Provider Service (RIPS). The role of Cache Manager is to store the frequently used and searched resource information in its own local cache in order to reduce the number of times Index Service is accessed. When some period of time has elapsed, however, we cannot guarantee whether the information in the cache is still up-to-date or not. Cache Auto Updater (CAU) module in Cache Manager is used to maintain the up-to-date resource information in the cache. CAU subscribes to RIPS on the resource site to receive a periodic notification of change in resource information.

After receiving query result from Cache Manager, IS Agent forwards this resource information to Job Information Parser through Query Processor. It is then combined with job specification in Job Information Parser and sent to Resource Selector.

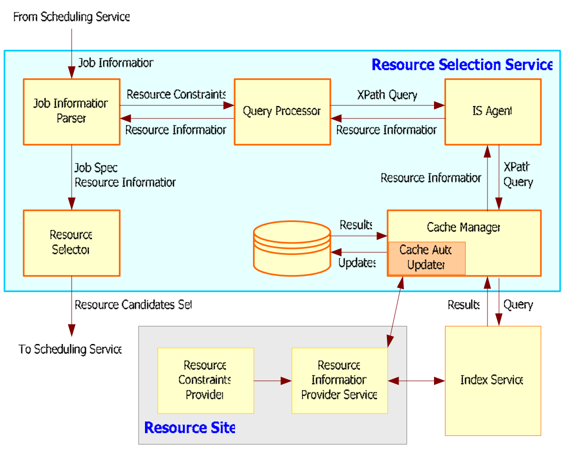

Figure 4. RSS Architecture

Resource Selector, the core module in RSS, selects resource candidates on which the job may be executed. As discussed in the previous section, Resource Selector considers both the job’s preferences on the resources (specified in job information) and the resource’s preferences on the jobs (specified in resource information). In other words, Resource Selector calculates jobPoint and resPoint from rank points of the job information and the resource owner policy. If a jobPoint or resPoint of a resource equal to zero, Resource Selector ignores the resource. After that, Resource Selector selects the resources with the higher jobPoint or resPoint and returns them to SS. After that, Resource Selector selects the number of resources (between min and max) among resources with the higher jobPoint or resPoint.

4.2. Scheduling Service (SS)

SS receives a list of resource candidates from RSS and assigns the job to one or more resources among the resource candidates. As figure 2(b) shows, each resource information returned from RSS presents jobPoint and resPoint. With these point values, SS can perform various scheduling. For example, SS can only consider one of the jobPoint and resPoint or both of them as the criteria to schedule the job. Currently, we have implemented a SS that considers both jobPoint and resPoint with the same ratio, and have used it in our experiments.

Furthermore, the schedulers may need more detail information about resources other than jobPoint or resPoint for more sophisticated scheduling algorithm.

For this, the schedulers can include <arguments>[3] part in the job information, requesting interesting attributes of the resources such as CPU power, available memory space, and available disk space, etc. After receiving the values of such attributes from RSS, SS can arrange the resources according to its own scheduling criteria to select the best target resource(s).

5. Experimental Results

5.1. Experimental Setup

Our experimental testbed consists of six 2.4GHz uniprocessor Pentium 4 nodes running Linux (cf. Figure 5). One node is dedicated to run our resource broker service and another node is used for generating job submission workloads. The remaining four nodes are used as worker nodes, on which the job is actually allocated and executed. One of the worker nodes additionally runs GT3 Index Service for aggregating host information from each worker node.

Applications used in this experiment perform simple arithmetic evaluation in a loop for 20 to 40 minutes. The actual running time follows a uniform distribution. This meaningless application is only used for introducing some CPU load during the running time. We also simulate the allocation of parallel jobs; a job may need a node count ranging from 1 to 4. If the node count is larger than 1, the same job is duplicated in the selected nodes.

5.2. RSS Performance

First, we have measured the performance of RSS. Table 1 summarizes the average execution time spent in each RSS component. We can see that most of times are spent for communicating with Index Service. This result effectively confirms our design choice that it is necessary to have Cache Manager to increase the performance of RSS and to reduce network traffics. If Index Services were located over wide area networks (WAN), the access cost would be increased much more substantially.

The actual advantage of using Cache Manager heavily depends on the cache hit rate. However, we are unable to get the meaningful data on the typical cache hit rate due to the small size of our testbed. It is interesting to study the performance of Cache Manager in heterogeneous, large-scale Grid environment, and we leave it for future work.

[3] Detail description of <arguments> is not included here due to space limitation.

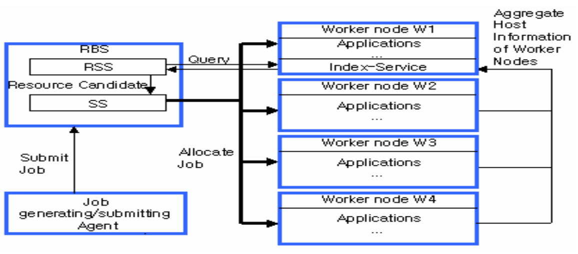

Figure 5. Experimental Setup

Table 1. Average Execution Time Spent in Each RSS Component

Module	Time(ms)
Query Generator	5.01
XML Parser	14.81
Query to Local Cache	59.87
Query to Remote IS	870.17
Resource Selector	4.25

5.3. Enforcing Resource Owner Policies

The purpose of next experiments is to test whether RSS enforces ROPs (resource owner policies) effectively during the resource selection phase. Currently, resource owners can specify their own policies based on the job owner, the job submission time, and the load level of the system. Each policy type is investigated independently in the following subsections.

5.3.1. Policies on the job owner

To test whether user policy is effectively reflected to the resource selection result, we have configured that each node has different user group-related policy. Job generating/submitting agent generates and submits 100 jobs with the randomly selected userSN and the node count. And then Resource Broker Service selects resources and allocates jobs to selected resources. UserSN is selected among "/O=Grid/O=Globus/OU=kaist.ac.kr" (KAIST) and "/O=Grid/O=Globus/OU=kisti.or.kr" (KISTI) and the node count is selected from 1 to 3. We have configured that W1 (W2) permit the jobs only from KAIST (KISTI) user group, and W3 allows all users to use it. In this experiment, ROP of W1 (W2) assigns the highest rank point to KAIST (KISTI), and ROP of W3 assigns 80 rank points to all users.

Figure 6 shows the distribution of jobs among three nodes. We can see that all jobs which are submitted from KAIST (KISTI) and require only one node, are allocated to W1 (W2). W3 is shared by all users according to the ROP of W3. Note that if a job requires

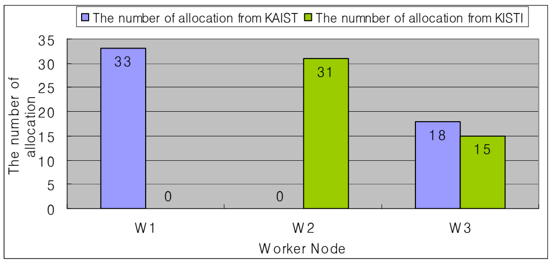

Figure 6. Number of Allocated Jobs to Each Worker Node

more than one node, the job can be allocated to nodes with higher resPoint.

5.3.2 Policies on the load level

If CPU load level becomes higher, resource owners may want to restrict other users to use their resources. Table 2 shows a scenario used in this experiment, where each node has different limit on the load level. For example, the worker node W2 accepts incoming jobs as long as its load level is less than 5. We have generated 400 job submission requests and their arrival rate follows a Poisson distribution with a mean of 1 request per minute.

Figure 7 shows the changes in the load level of worker nodes with respect to the elapsed time. We can see that the load level of W2, W3, and W4 does not significantly exceed the upper limit configured by the resource owner, while that of W1 increases continuously as W1 does not control the load level, increases continuously. The reason why the load level is not strictly kept under the upper limit is because the load level is not updated instantly in the Linux system (instead, it is calculated as an average over past 1 minute). The propagation delay of load information from a resource to Index Service and the time taken in allocating a job also contribute the deviation of the result from the expected value. If the inter-arrival time of job submission requests increases, RSS can select resources using relatively fresh information, so the error could be reduced.

5.3.3 Policies on the job submission time

Resource owners can also describe when other users can use their resources in ROP. Table 3 shows a service scenario, which has different policies on the job submission time. In this scenario, the resource owner of W2 simply does not want to offer the resource to Grid users during daytime (from 8:00 to 20:00), while W1 has no such restriction. In this experiment, we have

Table 2. Service Scenario Based on the Load Level

Node ID	Condition about load level	Rank point
W1	Default	10
W2	0~5	100
	5~	0
W3	0~10	100
	10~	0
W4	0~15	100
	15~	0

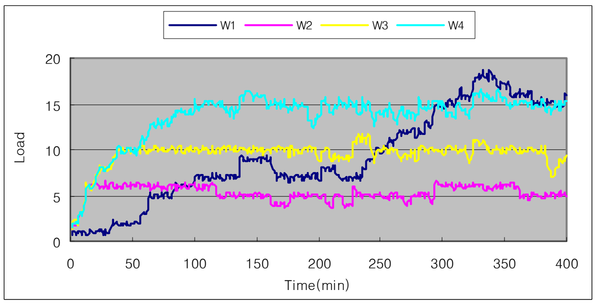

Figure 7. Differentiated Allocation Based on the Load Level

generated job submission requests whose arrival rate follows a Poisson distribution with a mean of 10 requests per hour.

Figure 8 presents the changes in the load level of W1 and W2 with respect to the elapsed time. It is obvious from the graph that no job has been allocated in W2 during daytime. Note that the load level is not decreased to 0 immediately at 8:00. This is because we assume the running jobs once allocated are not killed.

As W2 does not accept any job submission during daytime, we can see that all the jobs are allocated to W1, which increases the load level of W1 for the same period.

6. Conclusion and Future Work

In this paper, we have designed and implemented a new OGSI-compliant Grid resource broker service. Our resource broker service performs resource discovering and scheduling and hides the underlying complexity of Grid resources from Grid users. It can be easily extended to incorporate various resource scheduling services and other features as it supports a very general resource broker framework. Moreover, the proposed resource broker service considers resource owner policies as well as user requirements on the resources. Through experimental evaluations, we have successfully shown that the proposed resource broker service can effectively enforce resource usage policies

Table 3. Service Scenario Based on the Load Level

Node ID	Condition about time	Rank point
W1	default	100
W2	8:00~20:00	0
	the rest	100

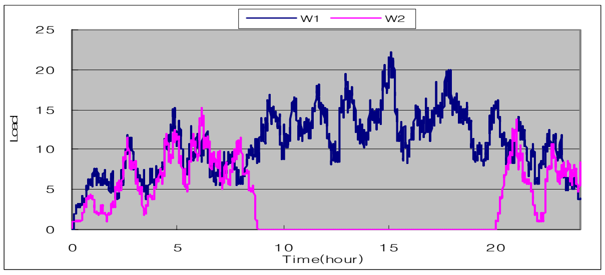

Figure 8. Differentiated Allocation Based on the Job Submission Time

based on the job owner, the job submission time, and the load level.

As experiments have been conducted on a rather small Grid testbed, however, it is necessary to investigate the scalability of the broker service on a much larger-scale Grid testbed. In addition, we plan to refine our broker service to minimize resource conflicts when parallel jobs are allocated.

7. References

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

[2] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and Its Implementation in the Nimrod-G Resource Broker," Future Generation Computer Systems, 18:1061-1074, 2002

[3] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, "Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI," IEEE Internet Computing, 2002.

[4] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration.", Open Grid Service Infrastructure WG, Global Grid Forum, 2002.

[5] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "Grid Services for Distributed System Integration," IEEE Computer, June 2002.

[6] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, T. Maguire, T. Sandholm, P. Vanderbilt, and D. Snelling, "Open Grid Services Infrastructure (OGSI) Version 1.0." Global Grid Forum Draft Recommendation, 6/27/2003.

[7] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuecke. "Security for Grid Services." Proc. 12th International Symposium on High Performance Distributed Computing, 2003.

[8] "Information Services in the Globus Toolkit® 3.0", http://www.globus.org/mds/

[9] "Resource Management : GT2 GRAM and GT3 GRAM", http://www-unix.globus.org/developer/resource-management.html

[10] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. "Grid Information Services for Distributed Resource Sharing." Proc. 10th International Symposium on High-Performance Distributed Computing, 2001.

[11] "Data Management Services", http://www-unix.globus.org/developer/data-management.html

[12] "GRAM RSL Schema Documentation", http://www-unix.globus.org/developer/data-management.html

[13]"GLUE Schema Activity", http://www.cnaf.infn.it /~sergio/datatag/glue/

[14] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid resource Management Systems", Software Practice and Experience, 32(2): 135-164, 2002.

[15] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", Proceedings of Supercomputing '96, 1996.

[16] D. Abramson, R. Buuya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker", Future Generation Computer Systems. 18(8), 2002.

[17] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids." Cluster Computing, 5(3):237-246, 2002.

[18] B. Chapman et al, "EZ-Grid Resource Brokerage System", http://www.cs.uh.edu/~ezgrid/ , 2001.

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