

eNANOS Grid Resource Broker

Ivan Rodero, Julita Corbalán, Rosa M. Badia, Jesús Labarta

CEPBA-IBM Research Institute
Technical University of Catalonia (UPC), Spain
{irodero, juli, rosab, jesus}@ac.upc.es

Abstract. Grid computing has been presented as a way to share geographically and organizationally distributed resources and to perform successfully distributed computation. For achieve this goals a software layer is necessary to interact with grid environments. Therefore, not only a middleware and its services are needed, it is necessary to offer resource management services to hide the underlying complexity of the Grid resources to Grid users. In this paper, we present the design and implementation of a general-purpose OGSI-compliant Grid resource broker compatible with both GT2 and GT3. It focuses in resource discovery and management and dynamic policies management for job scheduling and resource selection. The presented resource broker is designed in an extensible and modular way using standard protocols and schemas to become compatible with new middleware versions. We also present experimental results to demonstrate the resource broker behavior and performance.

1. Introduction

Grid computing [1] has emerged in last years as a way to share heterogeneous resources distributed over local or wide area networks and geographically and organizationally dispersed. It also provides the way to create virtual organizations (VOs) through secure resource sharing among individuals, institutions, and resources; it also helps to reduce resource replication. Grid computing builds on the concept of distributed computing, and software provides a way to divide up tasks so they are processed in parallel. In that context Grid computing is a good framework for solving large-scale problems typical of High Performance Computing (HPC) such as bioinformatics, physics, engineering or life sciences problems.

In order to provide the necessary infrastructure for the Grid several projects have been developed such as *Globus Toolkit* [2], Condor [3] or Unicore [4]. In particular, Globus Toolkit is being implanted in several projects, trying to provide a generic solution for a Grid infrastructure.

Recently Grid world, and more especially Globus Toolkit, is being involved in Web Services technologies, Globus Toolkit 3 (GT3) is based in *Open Grid Services Architecture* (OGSA) which uses a set of Grid Services to offer services needed for enabling Grid computing environment. OGSA defines what is called a *Grid Service* [5]: a Web service that provides a set of well-defined interfaces and that follows specific conventions. The interfaces address discovery, dynamic service creation, lifetime management, notification and manageability; the conventions address naming and upgradeability.

In addition to the infrastructure basic services to give support to paradigms like *Resource Management* [6] are also required. The resource management in Grid environments is different from the one used in cluster computing. At last, many efforts have been dedicated in HPC, especially in job scheduling policies and resource usage maximization. Globus Toolkit provides some useful services including Grid Security Infrastructure (GSI) [7], Grid Resource Allocation and Management (GRAM) [8], Data Management Services (e.g. gridFTP) [9], and Information Services, Monitoring and Discovery System (MDS) [10].

Discovering and selecting suitable resources for applications in Grid environments is still an open problem. Then, when a user wants to interact with a Grid, all processes related to resource management decisions should be handled manually. But these tasks are too difficult for a user and seem to be a good idea to take a *Resource Broker* or a meta-scheduler to do these basic functions. Additionally, no resource broker is included in top of the Globus Toolkit.

In this paper, we present the design and implementation of an OGSI-Compliant resource broker developed as a Grid Service. It is compatible with both Globus Toolkit 2 and Globus Toolkit 3 services, and implements flexible mechanisms to become compatible with next Globus versions. Our resource broker is centered in resource management and it focuses in *dynamic policy management*. This resource broker is responsible of the Resource Discovery and Monitoring, Resource Selection, Job Submission and Job Monitoring; and implements policy management mechanisms from user side. It supports different policy classes including scheduling policies, resource selection policies and complex policies (called meta-policies). It uses a XML based language to specify user multi-criteria. It also provides a set of Grid Services interfaces and Java API for various clients, e.g. user applications, command-line clients or grid portals.

The rest of this paper is organized as follows. Section 2 overviews previous research on resource brokering and scheduling. Section 3 discuss the system design and implementation details of our Grid resource broker. Section 4 describes experimental results and section 5 concludes the paper and presents future work.

2. Related Work

At the moment there are many projects related to Grid since it is an important research issue for the international community. Some projects, such as AppLes [11], Nimrod/G [12], Condor-G [13], EZ-Grid [14], GridLab Resource Management System (GRMS) [15] or GridWay [16], have been working on brokering systems. These projects are developed on top of GT2 but other initiatives have been presented, for instance a design and implementation of a Grid Broker Service in terms of OGSA [17] running on GT3.

Our Grid resource broker differs from previous existing brokerage systems in the following aspects: First, this general-purpose resource broker is compatible with GT2 and GT3 services, it implies that is needed a uniform internal representation of objects and data involved in any task of resource management; secondly, the proposed resource broker provides dynamic policy management which combined with user multi-criteria requirements allows to advanced users a large capacity of decision. This

user multi-criteria file is a XML document and can be used in policies evaluation and is composed of requirements and recommendations. A requirement (hard attribute) is a restriction for resource filtering and a recommendation (soft attribute), “with its priority,” can be used to have a resource ranking for policies evaluation. Finally, since our resource broker is implemented as a grid service, we can have several broker instances in the same or different machines and construct more scalable systems.

3. System Design and Implementation

3.1. Overall Architecture

This subsection presents the overall architecture of the proposed Grid resource broker. As shown in Fig. 1, the broker consists of five principal modules, a queuing system and data system for persistency. Moreover, the system is composed of Globus Toolkit services and an API to access the broker services.

Resource Discovery uses both GT2 MDS (GRIS/GIIS servers) and GT3 Information Services (based in Web Services). It uses a uniform representation of resource servers and resources based on GLUE schema.

Resource Selection performs dynamic selection of best resources from job specifications, user criteria, resource information and policies evaluation. All decisions related to resources are made from the local data obtained in resource discovery and monitoring processes.

Resource Monitoring gathers information about resources and stores it as local information which is available in “real-time” for broker modules and users.

Job Submission performs job submission to GT2 or GT3 systems depending of user criteria and job characteristics. It receives a user criteria and RSL from the user side. To select the appropriate job from local queues the scheduling policy is evaluated.

Job Monitoring controls job status changes and stores their history. It also performs job rescheduling when appropriate (e.g. when a resource has fallen). To do that, some interactions between resource monitoring and job monitoring is needed.

The API is the responsible to provide a unique point of access to broker services. This API can be used by different clients such as user applications, grid portals or command-line.

The broker design is based on Globus Toolkit as a middleware and as the provider of basic services. Also, the design is enough extensible to make easy to adapt the broker to new Globus versions. In order to obtain this, uniform

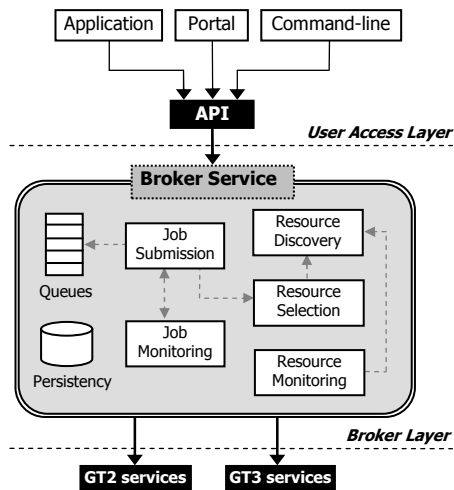


Fig. 1. Overall architecture

and standard schemes have been used (e.g. GLUE based schema is used for internal resource representation). Recently, some Globus versions have appeared and it is not clear how the evolution of the Grid technology will be. At this time, the Globus project is working in implementations based in Web Services technology, e.g. Web Service Resource Framework (WSRF). These new technologies can be very useful but is very important to keep the compatibility with systems based in previous Globus versions and to give support to its users. There are a lot of projects related in different topics developed on top of GT2, e.g. DataGrid [18], GridLab [15] or GRID SuperScalar [19]. More detailed description of our broker architecture is presented in the following subsections and more information can be found in [20].

3.2. Job Description and User Criteria

To describe a job a RSL is required and a user criteria is optional. We do not extend RSL schema in order to simplify files and separate concepts. A user criteria is XML-based and specifies basic parameters. A user criteria is composed of several attributes organized in three categories: Memory&Processor, Filesystems&OS, and Others. Each attribute is composed of various elements as is shown in Fig. 2:

- *Name*: name of the attribute (e.g. RAMAvailable, ClockSpeed, OSName, etc.)
- *Type*: attributes values can be STRING or INTEGER
- *Operator*: if the attribute type is STRING the possible operator is “==” (identical strings) and if it is an INTEGER attribute possible operator are “==”, “<=” or “>=”
- *Value*: value of the attribute (corresponding to its type)
- *Importance*: There are two types of attributes, HARD and SOFT attributes. A HARD attribute is a requirement for resources and must be accomplished. However, a SOFT attribute is a recommendation for choosing between all resources that accomplish their requirements.
- *Priority*: this element is considered only in SOFT type attributes in order to obtain a ranking of resources according to the user criteria. The obtained rank value can be useful for later policies evaluation.

```
<?xml version="1.0" encoding="UTF-8"?>
<CRITERIA>
  <Memory-Processor>
    <Attribute Name="RAMAvailable" Operator="&gt;=" Value="100" Type="INTEGER" Importance="HARD" Priority="1" />
    <Attribute Name="VirtualAvailable" Operator="&gt;=" Value="250" Type="INTEGER" Importance="SOFT" Priority="3" />
    <Attribute Name="ClockSpeed" Operator="&gt;=" Value="500" Type="INTEGER" Importance="SOFT" Priority="7" />
    <Attribute Name="LoadLast15Min" Operator="&lt;=" Value="45" Type="INTEGER" Importance="SOFT" Priority="10" />
  </Memory-Processor>
  <FileSystem-OperatingSystem>
    <Attribute Name="AvailableSpace" Operator="&gt;=" Value="600" Type="INTEGER" Importance="SOFT" Priority="7" />
    <Attribute Name="OS Name" Operator="==" Value="Linux" Type="STRING" Importance="HARD" Priority="1" />
  </FileSystem-OperatingSystem>
  <Others>
    <Attribute Name="Total CPUs" Operator="&gt;=" Value="4" Type="INTEGER" Importance="SOFT" Priority="1" />
    <Attribute Name="MaxQueueTime" Operator="==" Value="3600" Type="STRING" Importance="SOFT" Priority="1" />
  </Others>
</CRITERIA>
```

Fig. 2. User criteria example

3.3. Policy Management

As well as basic brokering functions (resource discovery, job submission, etc.) dynamic management of policies and the implementation of the necessary mechanisms to support them, are important subjects in the design of this broker. The selection of the better job and better resource for a given configuration is an optimization problem with NP-Complete solution. In order to reduce and divide the complexity, the broker works with two kinds of basic policies, one for a job scheduling and another for resource selection. Also, above job scheduling and resource selection policies, a meta-policy is offered, which can be implemented with genetic algorithms or other optimization methods. The evaluation process of policies is shown in Fig. 3 and consists of three phases. First an initial evaluation of the job scheduling policy is performed and then, for each job selected, the resource selection policy is evaluated and finally the meta-policy evaluation is performed. A meta-policy evaluation consists of choosing the best job to be executed and the best resource for the execution from the data structure obtained of the evaluation of the previous policies. This data structure is a matrix corresponding of the set of jobs obtained in the first step and for each of them a set of resources obtained in the second one.

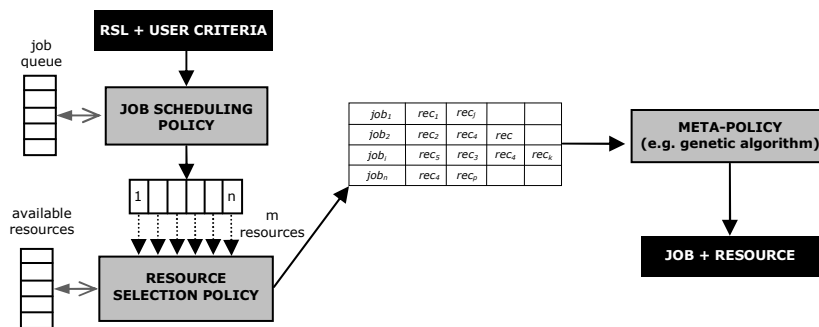


Fig. 3. Policies evaluation schema

Some policies implementations for the broker are exposed next. For job scheduling `FIFOjobPolicy` (First In First Out), `REALTIMEjobPolicy` (minimizes `REALTIME=deadline time-estimated time of job finalization`), `EDFjobPolicy` (Earlest Deadline First). For resource selection `RANKresPolicy` (resource selection based in the greatest rank obtained from the resource filtering process), `ESTresPolicy` (Earlest Starting Time, based in the estimated waiting time for a job in a local queue). User criteria can be used in some evaluation policies like `RANKresPolicy`.

In order to obtain dynamic policy management we propose a design based in generic interfaces for each kind of policy. Then, the mechanism is ready for a policy evaluation independently of its implementation.

Dynamic management of policies allows the user to manage them. Considering that several instances of the broker can exist, it is possible to have at the same time broker instances with different policies. To manage policies from the users side some interfaces are available to see the established policy, to change it and to see what policies are available.

3.4. Job Scheduling and Management

When a job is submitted is automatically queued in the local system. Periodically the resource broker tries to schedule all jobs according to the established policies. When a machine fails, all the jobs running in that machine are rescheduled through another local queue called retrying. This queue is of higher priority than the submit one in order to avoid inanition situations. Any submitted job is scheduled until all jobs from retrying queue are managed.

The main issues of job management are job submission, cancellation, monitoring and termination. In order to submit a job, a RSL is required and optionally a user criteria file. The RSL can be a traditional RSL or XML based RSL-2 because the resource broker is compatible with both GT2 and GT3. If the RSL used is the traditional one this job could be executed in a GT2 or GT3 resource indifferently. If RSL-2 is used, the job can only be executed in a GT3 resource because no RSL-2 to RSL parser exists yet. Details about user criteria are shown in subsection 3.2.

To submit a job to a certain resource the Globus GRAM API is used. Callbacks are managed with the GRAM interface responsible of status changes. To decide which is the appropriate resource for a job execution the resource selection policy is evaluated over the resources obtained previously from the resource discovery module.

In the job monitoring process the resource broker is looking for notifications and callbacks to control the job status. Also the job history is kept to know what is happening and what happened during the job life. In job history some information is considered like date, time, operation and other details. In order to preserve data persistence of submitted jobs a recovery file is saved with the necessary job information to resubmit them. In case that the resource broker machine crash, when the resource broker is restarted, all jobs in recovery file are rescheduled to be executed.

In the implementation of this resource broker we had some problems related with the Globus APIs, in particular with GRAM. Globus infrastructure adds overflow in job submission and GRAM interfaces are not compatible with different Globus versions at the same time. Then, we had to implement different interfaces and objects in the job management to give support to GT2 and GT3. Furthermore Globus client APIs are thought to be used only from the final user side and we had some problems in job submission from a Grid Service. We had to make some changes in code and correct some bugs.

3.5. Resource Management

Our resource broker has a generic and unique representation of resources based in GLUE schema. Therefore, we can use only one internal representation for GT2 and GT3 resources and we can make some decisions independently of the Globus version of resources. The main attributes for this resource representation are general information such as Globus version, hostname or #CPUs, main memory info, operating system, processors info, processors load, file systems info and running jobs.

In order to simplify the resource discovery process we used a uniform representation for resource servers called *Global Grid Resource Information Server* (GGRIS). In this representation we can specify a MDS GIIS, a GRIS or a GT3 Index

Service. From these resource servers the resource broker can obtain resources and resources details in the previous shown representation.

Resource information is dynamic and the only required functionality is to maintain persistent is the GGRIS information. Due to this, we use a XML file with a list of available resource servers. Depending of the server type, different information is needed for specifying its location. For GT3 servers only is needed the Index Service GSH location. However, for GT2 GRIS or GIIS servers, the hostname, port and baseDN are needed.

Resource monitoring updates local data about resources calling the resource discovery module continuously. In order to detect when a resource has failed the resource broker compares current available resources with the previous data before updating the list of resources. In the case of a detection of one or multiple resource fallen, this module interacts with job management modules rescheduling their jobs.

In both GT2 MDS and GT3 Index Service we use the scripts provided by Globus. The GT3 Index Service is a useful mechanism for indexing data but the scripts provided by Globus are not enough powerful in some cases. In general, there is a lack of information about local information of resources. For instance, the behavior of applications is very useful information to make scheduling decisions in global terms. Consequently, is difficult to give good support to HPC resources with Globus infrastructure.

4. Experimental Results

Our experimental testbed consist of various heterogeneous resources from three organizations: CEPBA, UPC-DAC, and UCM-DACYA. We dispose some machines from a powerful IBM RS-6000 & IBM p630 cluster to a simple laptop.

```

pcirodero:~/test$ get_AllJobs ①
All submitted jobs:
There are any job !

pcirodero:~/test$ job_submit rsl1.xml criteria1.xml
Job submitted successfully with id: 1@1087831803184
pcirodero:~/test$ job_submit rsl2.xml criteria2.xml
Job submitted successfully with id: 2@1087831807889 } ②

pcirodero:~/test$ get_AllJobs
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_PENDING
Job ID: 1@1087831873835 at status: JOB_PENDING ← ③

pcirodero:~/test$ add_ggris http://pcirodero.ac.upc.es:.../IndexService
GGRIS added successfully: pcirodero.ac.upc.es

pcirodero:~/test$ get_AllJobs
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_PENDING ④
Job ID: 1@1087831873835 at status: JOB_PENDING

pcirodero:~/test$ get_AllJobs
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_SUBMITTED ⑤
Job ID: 1@1087831873835 at status: JOB_SUBMITTED

pcirodero:~/test$ get_AllJobs ← ⑥
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_RETRYING ⑦
Job ID: 1@1087831873835 at status: JOB_RETRYING

pcirodero:~/test$ add_ggris http://pcmas.ac.upc.es:.../IndexService
GGRIS added successfully: pcmas.ac.upc.es

pcirodero:~/test$ get_AllJobs ← ⑧
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_RETRYING
Job ID: 1@1087831873835 at status: JOB_SUBMITTED ← ⑨

pcirodero:~/test$ job_history 1@1087831873835
21/5/2004 17:30:3 =>JOB CREATION
21/5/2004 17:30:3 =>JOB QUEUED in PENDING queue
21/5/2004 17:31:13 =>RESTORED From Recovery File
(to be created another time)
21/5/2004 17:31:13 =>JOB CREATION
21/5/2004 17:31:13 =>JOB QUEUED in PENDING queue
21/5/2004 17:31:52 =>JOB SUBMITTED to pcirodero.ac.upc.es
21/5/2004 17:33:1 =>JOB QUEUED FOR RETRYING because
resource pcirodero.ac.upc.es has down
21/5/2004 17:34:2 =>JOB SUBMITTED to pcmas.ac.upc.es ← ⑩

pcirodero:~/test$ get_AllJobs
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_SUBMITTED
Job ID: 1@1087831873835 at status: JOB_DONE ⑪

pcirodero:~/test$ get_AllJobs
All submitted jobs:
Job ID: 2@1087831873909 at status: JOB_DONE ⑪
Job ID: 1@1087831873835 at status: JOB_DONE

```

Fig. 4. Execution of some broker commands

4.1. Behavior Analysis

In order to illustrate the broker behavior we are going to use an example shown in Fig. 4. In that example we can find several circumstances and actions related with the broker. Next, we explain each event and what decisions takes the broker system.

Initially, no resource is available and no job is submitted. Next in (2), some jobs are submitted to the broker and are queued to the pending queue. In (3) the broker falls. When the broker is restarted we add a computational resource in order to execute submitted jobs. Then, in (4) the broker retrieves previous submitted jobs from the recovery system and queue them again in local queue. In (5) jobs are submitted to the available resource and begin their execution. Suddenly, the resource which was executing jobs (pciroadero) falls, in (6). So, in (7) all jobs that were submitted in pciroadero are queued to retry their execution. Now we do not have any resource available, then in (8) we add a new computational resource to allow job execution. Afterwards jobs begin their execution on pccmas, in (9). In (10) we can see a job history and how all events have happened. Finally, in (11) all jobs finish their execution on pccmas.

4.2. Performance Analysis

In order to study the broker and system performance, we are instrumented the broker through JIS and JACIT [21]. JIS allow us to instrument Java classes and obtain some traces which can be visualized and analyzed with Paraver [22]. First, we will expose results obtained from an execution of a minimum job in a GT3 resource. So, with the results we can approximate various types of overhead such as Globus, the broker or communications overflow. In Fig. 5 a trace obtained from the execution of two minimal jobs is shown. In Paraver traces we can see the time in X axis, each row represents a thread, colors represent different states and flags are events. We can see some events (such as submission requests, job dispatching actions, finalization notifications and state requests) or the spent time for each one.

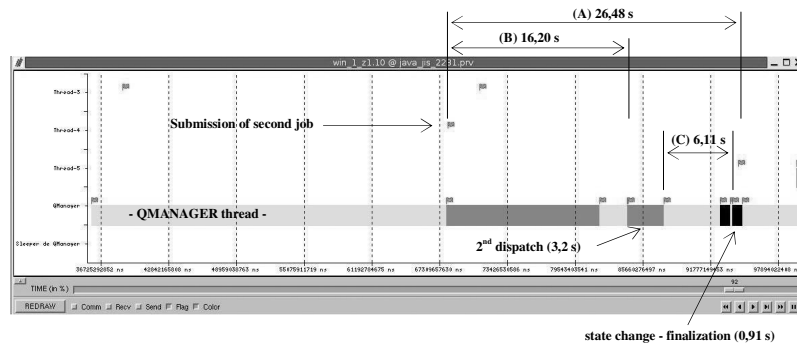


Fig. 5. Trace obtained in the minimal execution

- (A) is the elapsed time between the reception of the job submission and the moment of its conclusion (including the state change, total process).
- (B) is the queuing time of the job until its submission to a specific resource.

(C) is the elapsed time between the job submission to a specific resource and the notification of its conclusion.

Then, considering the data obtained in this test, we can get some numbers related to the overhead. Broker overhead=61%, globus overhead=16%, other overhead is not relevant. **Total overhead=77%** but this is only the result obtained from a concrete execution of a minimal job.

Now we are going to present average results obtained from the execution of several tests of different duration. In Fig. 6 is shown results of that arithmetic jobs, where executions where short executions have big overhead but from job of a minute duration, we obtain really acceptable values. Then we can say the broker is enough suitable for Grid oriented applications¹.

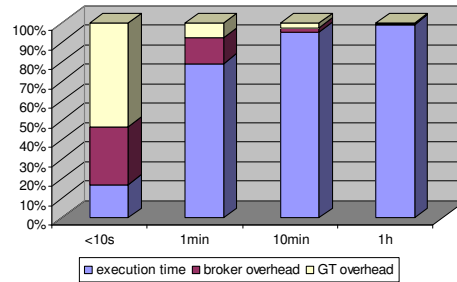


Fig. 6. Distribution of time in tests

5. Conclusion and Future Work

In this paper, we have designed and implemented an OGS-compliant Grid resource broker. Our resource broker performs resource discovery and management, scheduling and hides the underlying complexity of Grid resources from Grid users. It is compatible with both GT2 and GT3 services and is designed as an extensible and modular way to be easily extended and become compatible with future Globus versions. Moreover, the proposed resource broker considers dynamic policies management. For achieve this goals, the resource broker implement powerful mechanisms to allow users manage policies using a Grid Service based interface, API, and client. Through experimental evaluations, we have successfully shown the resource broker system behavior and its performance is satisfactory for Grid oriented applications.

As future work we are going to improve our resource broker with more robustness, check pointing, job migration and so on. We plan to add low level interaction with local queuing systems in order to choose the best approaches in Grid environments especially for parallel applications. Also, we need to implement complex meta-policies and new policies based in prediction concepts. We want to implement support for GT4 when this is stable. Finally, we plan to construct more scalable systems.

With our experience developing a broker on top of Globus Toolkit, we have found some deficiencies. First, we think it would be need to improve APIs to give better support for developers, currently Globus APIs are thought to be used for final users. Resources should be managed in a more effective way. A useful middleware should provide good monitoring tools and enable communications between the middleware and the local environment. Finally, the Globus Toolkit should be improved to reduce its overhead; we hope future versions will be better.

¹ In this paper we do not consider results from the overhead of data transport and management

6. Acknowledgments

This research has been supported by the Ministry of Science and Technology of Spain under contract TIC2001-0995-C02-01, and the European Union project HPC-Europa under contract 506079.

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. "The Globus Project (Globus Alliance)", <http://www.globus.org>
3. M. Litzkow, M. Livny, and M. Mutka, "Condor – A Hunter of Idle Workstations", *Proceedings of the 8th International Conference of Distributed Computing Systems (ICDCS 1988)*, January 1988, San Jose, CA, IEEE CS Press, USA, 1998
4. "Unicore Project", <http://www.unicore.org>
5. 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.
6. Jarek Nabrzyski, Jennifer M. Schopf, and Jan Weglarz, "Grid Resource Management, State of the Art and Future Trends", *Kluwer Academic Publishers*, 2004
7. "Globus Security Infrastructure", <http://www.globus.org/security>
8. "Resource Management: GT2 GRAM and GT3 GRAM", <http://www-unix.globus.org/developer/resource-management.html>
9. "Globus Data Management Services", <http://www-unix.globus.org/toolkit/docs/3.2/datamanagement.html>
10. "Information Services in the Globus Toolkit", <http://www.globus.org/mds>
11. F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", *Proceedings of Supercomputing'96*, 1996
12. 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(8), 2002
13. J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids". *Proceedings of the Tenth International Symposium on High Performance Distributed Computing*, IEEE CS Press, August 2001
14. B. Chapman et al, "EZ-Grid Resource Brokerage System", <http://www.cs.uh.edu/~ezgrid/>
15. "GridLab, A Grid Application Toolkit and Testbed", <http://www.gridlab.org>
16. "The GridWay Project", <http://asds.dacya.ucm.es/GridWay/>
17. Young-Seok Kim, Jung-Lok Yu, Jae-Gyoon Hahm, Jin-Soo Kim, and et al. "Design and Implementation of an OGSi-Compliant Grid Broker Service", *Proc. of CCGrid 2004*
18. "The DataGrid Project", <http://www.eu-datagrid.org>
19. Rosa M. Badia, Jesús Labarta, Raül Sirvent, Josep M. Pérez, José M. Cela, and Rogeli Grima, "Programming Grid Applications with GRID superscalar", *Journal of Grid Computing*, January 2004
20. Ivan Rodero, Julita Corbalán, Rosa M. Badia, Jesús Labarta, "Providing a Resource Broker for eNANOS Project", *UPC-DAC 2004*
21. Jordi Guitart, Jordi Torres, Eduard Ayguadé, José Oliver, and Jesús Labarta, "Java Instrumentation Suite: Accurate Analysis of Java Threaded Applications", *2nd Annual Workshop on Java on High Performance Computing*, Santa Fe, New Mexico, USA, 2000.
22. Paraver, <http://www.cepba.upc.edu/paraver/>