Computing and Informatics, Vol. 27, 2008, 187–197

CROSSBROKER: A GRID METASCHEDULER FOR INTERACTIVE AND PARALLEL JOBS

Enol Fernández, Andrés Cencerrado
Elisa Heymann, Miquel A. Senar

Departament d'Arquitectura de Computadores i Sistemes Operatius
Universitat Autònoma de Barcelona
Barcelona, Spain
e-mail: {enol@caos., acencerrado@caos., elisa.heymann@, miquelangel.senar@}uab.es

Abstract. Execution of parallel and interactive applications on a Grid environment is a challenging problem that requires the cooperation of several middleware tools and services. In this paper, we present our experiences in the development of CrossBroker, a job management service that provides transparent and reliable support for such types of applications. We outline the main components of CrossBroker and how they interact with other middleware services. We also describe specific features of the scheduler used to guarantee resource co-allocation for running MPI jobs remotely over multiple machines spread across several Grid sites or to start interactive applications as fast as possible. These features include a simple time-sharing mechanism that allows fast execution of interactive applications even under heavy occupancy of Grid resources.

Keywords: Grid scheduling, MPI, interactive

1 INTRODUCTION

Grid environments constitute one of the most promising computing infrastructures for computational science. Large-scale Grid computing requires a job management service that addresses new concerns arising in the Grid environment. This job management involves all aspects of the process of locating various types of resources, arranging these for use, utilizing them and monitoring their state. In these environments, job management services have to deal with a heterogeneous multi-site computing environment that, in general, exhibits different hardware architectures, loss of centralized control, and as a result, inevitable differences in policies. Additionally, due to the distributed nature of the Grid, computers, networks and storage devices can fail in various ways. Most systems described in the literature follow a similar pattern of execution when scheduling a job over a Grid. There are typically three main phases as described in [1]: 1) resource discovery, which generates a list of potential resources to be used; 2) information gathering on those resources and the selection of a best set; and 3) job execution, which includes file staging and cleanup.

Many Grid initiatives follow these scheduling phases by providing the middleware infrastructure to develop applications on computational grids and to manage resources. The job management system that we have developed follows the same approach in scheduling jobs. However, our system, known as CrossBroker, is targeted to some applications that have received very little attention to date. Most existing systems have focussed on the execution of sequential jobs, the Grid being a large multi-site environment where jobs run in a batch-like way. CrossBroker also supports interactive jobs and parallel applications mostly written with the MPI library, being able to take advantage of being executed on multiple grid sites.

From the scheduling point of view, support for parallel applications introduces the need for co-allocation. In the case of interactive applications, the main requirement is the possibility of starting the application in the immediate future, also taking into account scenarios in which all computing resources might be running batch jobs. Interactive applications also require the ability to control application output online and the ability to change the execution parameters while the application is running.

The rest of this paper is organized as follows: Section 2 briefly describes the overall architecture of our Job Management services, Section 3 deals with the specific mechanisms available in CrossBroker for multi-programming, Section 4 describes the particular services that support submission of Interactive jobs, Section 5 describes the CrossBroker support for parallel applications on a grid, and Section 6 summarizes the main conclusions of this paper.

2 DESCRIPTION OF THE CROSSBROKER ARCHITECTURE

The architecture of the CrossBroker is tightly related to the software architecture of CrossGrid [2] and int.eu.grid [3] projects. We restrict our discussion here only to those elements that are directly related to the execution of applications. When users submit their application, our scheduling services are responsible for optimizing scheduling and node allocation decisions on a user basis. Specifically, they carry out three main functions:

1. Select the "best" resources that a submitted application can use. This selection takes into account the application requirements needed for its execution, as well as certain ranking criteria used to sort the available resources.

2. Perform a reliable submission of the application onto the selected resources. This involves the proper co-allocation of resources when the application is distributed among multiple sites.
3. Monitor the application execution and report on job termination.

Figure 1 presents the main components that constitute our resource management services. Each grid site is composed of a cluster of machines consisting of a gatekeeper, called Computing Element (CE), and many worker nodes (WN) managed through a Local Resource Manager, such as PBS or Condor. A user submits a job to a Scheduling Agent (SA) through a User Interface (command line or Migrating Desktop). The job is described by a JobAd (Job Advertisement) using the EU-Datagrid Job Description Language (JDL) [4], which has been conveniently extended with additional attributes to reflect the requirements of CrossBroker applications.

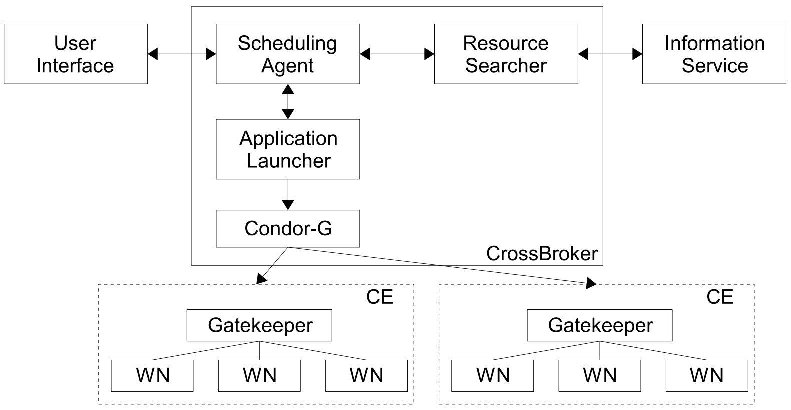


Fig. 1. CrossBroker resource manager architecture

Once the job has reached the SA, the Resource Searcher (RS) is asked for resources to run the application. The main duty of the RS is to perform the matchmaking between job needs and available resources. Using the job description as input, the RS returns as output a list of possible resources within which to execute the job taking into account the job requirements. The matchmaking process is based on the Condor ClassAd library [5], which has been extended with a set matchmaking capability [6]. The information on the status of each remote site is obtained through an information service built with Globus MDS[7] or RGMA.

The SA then selects the best resource (or group of resources) from the list returned by the RS. The computing resources (or group of resources) are passed to the Application Launcher, which is responsible for the actual submission of the job. Due to the dynamic nature of the Grid, the job submission may fail on that particular site. Therefore, the Scheduling Agent will try other sites from the returned list until the job submission either succeeds or fails. The Application Launcher is also in charge of the reliable submission of parallel applications on the Grid.

2.1 Job Description Language

As stated previously, the user describes the applications using a JDL file. Figure 2 depicts an example of such file for an interactive and Open MPI application.

```
Executable                = "fusion_app";
Arguments                 = "-n";
JobType                   = "Parallel";
SubJobType                = "openmpi";
NodeNumber                = 12;
Interactive               = True;
InteractiveAgent          = "i2glogin";
InteractiveAgentArguments = "-p 193.159.108.145 -t -c"
InputSandBox              = {"fusion_app"};
```

Fig. 2. JDL job description

The *Executable* and *Arguments* attributes specify the name and arguments of the application to be executed, while the *NodeNumber* attribute allows users to specify how many nodes their application will run on. The *InputSandBox* attribute is used to state the list of input files required for executing the application (in this example only the executable file will be transferred). The *JobType* attribute specifies the type of application (sequential/parallel/workflow[8]). An additional *SubJobType* specifies the kind of parallel application and application starter used; additional attributes for controlling the matchmaking and start of MPI applications are described in Section 5. The *SubJobType* values currently supported are 1) *openmpi*: defines an application linked with Open MPI; 2) *mpich*: an application linked with the MPICH library (ch_p4 device); 3) *pacx-mpi*: an application that can run over multiple sites with PACX-MPI[9]; 4)*mpich-g2*: inter-cluster application using the MPICH-G2 library[10]; and 5) *plain*: an application that does not use any of mpi implementations above, the CrossBroker will allocate the nodes and the user is responsible for its usage.

Independently of the type of application, the *Interactive* attribute defines the job as interactive and triggers the special scheduling mechanisms for such jobs. In order to provide real interactivity, application input has to be forwarded from the users (working on the executing machine) and their application (running on different nodes of the grid). The output must follow the reverse path. The CrossBroker uses a split execution system composed of two software components: an agent and a shadow. The agent traps the input and output of an application, and forwards them to a shadow process on another machine via the network. Under this arrangement, a program can run on any networked machine and still execute exactly as if it were running on the same machine as the shadow. The *InteractiveAgent* and *InteractiveAgentArguments* attributes allow the user to specify which agent to use. Section 4 specifies the interactivity features of CrossBroker with more detail.

3 SUPPORT FOR MULTI-PROGRAMMING

The CrossBroker has a time-sharing mechanism that enables both interactive and batch jobs to share a single machine, in such a way that the interactive application starts its execution as soon as it is submitted (unless all resources in the Grid are busy executing other interactive applications) and proper co-allocation is ensured. This time-sharing scheme takes advantage of the Condor Glide-In [13] mechanism, which we extended and adapted to meet our requirements. The main idea consists of submitting a job agent. The agent gains control of remote machines independently of the local-site job manager. Each machine acquired by our agent is configured as two virtual machines, in order to create a separate group of dedicated resources for two types of application: batch, on the one hand, and interactive, on the other. It is worth noting that our concept of virtual machines is a lightweight view and does not correspond to the classic view of virtual machines [14] that presents the image of multiple operating system configurations – completely isolated from each other – sharing a single machine. In our case, the machine only runs one O/S, but we split the machine into two separate execution slots. Each slot contains the executable and files required by the corresponding job. From the logical point of view, batch jobs will run on one virtual machine and interactive jobs will run on the other. However, our agent guarantees that interactive jobs will be executed at a higher priority than batch jobs. When the interactive job is finished, the original priority of the batch job is restored and after completion of the batch job, the agent leaves the machine.

Figure 3 shows an example machine that is logically divided into two virtual machines, one for executing a batch job and another for executing an interactive job (which may be either sequential or a job belonging to a parallel application). If there are more batch jobs to be executed than resources available, such batch jobs are queued until they can be executed.

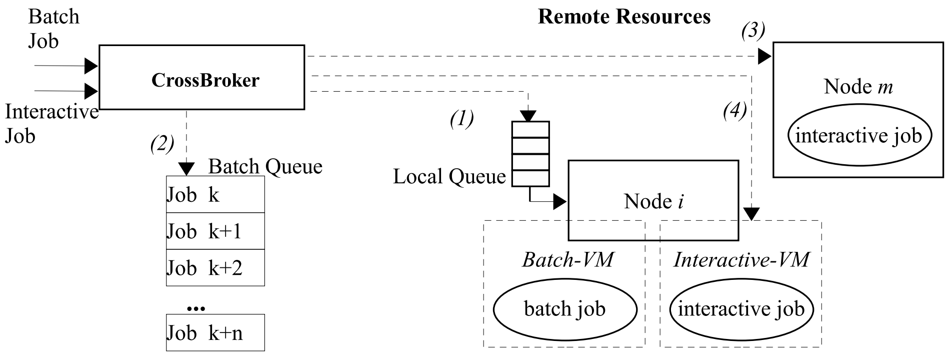


Fig. 3. Multi-programmed execution of interactive and batch applications

To summarize, there are different scenarios when submitting interactive or batch applications on the grid:

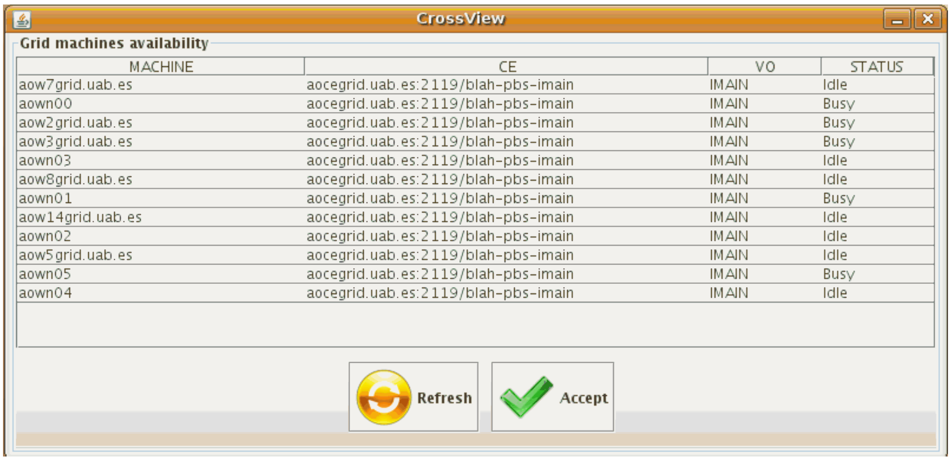
A batch application will trigger the execution of an agent if there is either any machine available or space in the queues managed by the local scheduler. Once started, the agent will create two virtual machines on the Worker Node(s): one for batch jobs (batch-vm) and another for interactive jobs (interactive-vm). The batch job will start its execution on the virtual machine devoted to batch applications (batch-vm). This situation is illustrated by arrow (1) in Figure 3. If there are no resources available, the application is queued in the CrossBroker to wait for a batch virtual machine to become idle. This situation is illustrated by arrow (2) in Figure 3.

For interactive applications, the CrossBroker will try to allocate a free machine to execute them. If there are CPUs available that meet the job requirements, the job will be submitted there without any agent, to enable rapid start-up and execution of the application. This situation is illustrated by arrow (3) in Figure 3. If there are not enough idle machines for executing the interactive job, the CrossBroker will search for interactive virtual machine available (interactive-vm). The job will be sent to one of the virtual machines, causing the batch job executing on the other virtual machine to lower its priority so as to benefit the interactive job. This situation is illustrated by arrow (4) in Figure 3. In addition, it is possible to have a combination of machines with and without agents for executing a parallel interactive application (combining cases 3 and 4). If there are not enough machines (with or without agents) to execute the application, the submission of an interactive application will fail. An interactive application will never pre-empt another already-running interactive application.

This mechanism improves resource availability for interactive jobs that will be able to run even under the circumstances of high Grid-resource occupancy. On the other hand, this has little impact on batch jobs that will undergo some execution delay when sharing their CPU with interactive jobs. However, given the nature of batch jobs, this delay is not particularly problematic and its impact is compensated for by the more moderate worsening of user priority.

Obviously, the OS overhead of multi-programming incurs a certain cost, but assuming there is adequate physical memory for all running jobs, this should be minimal. In fact, our multi-programming system could be used to initially create more than two virtual machines or to create new virtual machines dynamically. For instance, a larger degree of multi-programing may help overall throughput by running one batch job and several interactive jobs on the same machine.

Moreover, a specific tool has been developed in favor of the users using the CrossBroker. This tool, called CrossView, merges the information provided by the Information Service and the information of virtual machines directly managed by the CrossBroker. The users can check what resources are available at a given moment and select those resources for executing their jobs. Figure 4 shows a screenshot of the CrossView where the list of current virtual machines can be seen.



MACHINE	CE	VO	STATUS
aow7grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aown00	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Busy
aow2grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Busy
aow3grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Busy
aown03	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aow8grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aown01	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Busy
aow14grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aown02	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aow5grid.uab.es	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle
aown05	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Busy
aown04	aocegrid.uab.es:2119/blah-pbs-imain	IMAIN	Idle

Fig. 4. CrossView screenshot with resource availability information

4 MANAGING INTERACTIVE JOBS

One of the main requirements for the application to be interactive is to provide a possibility for starting it in a predictable future. This requirement is very hard to fulfil in most existing Grid infrastructures, due to their batch-oriented system of submission. The CrossBroker, in contrast, includes particular features to support interactive-oriented submission of jobs.

There are several mechanisms used by the CrossBroker in order to provide a high resource utilization for batch jobs and a low response time for interactive jobs. The most relevant mechanisms are the following:

- On-line scheduling. The scheduler tries to run each interactive job immediately. The job will be sent to any of available resources at submission time. Due to the lack of perfect synchronization between the status of each resource and the information published about them, some jobs may enter a queue rather than start executing immediately. In such a case, the scheduler will resubmit the job to any other resource from the list of available ones.
- Job multiprogramming mechanism. As stated in Section 3, when no free resources are available at any given time, an interactive job is submitted to the same machine where a batch job is already running and start immediately.
- Exclusive temporal access to resources. Resource selection may take place concurrently between several jobs that arrive at the Scheduler simultaneously. The exclusive temporal access mechanism guarantees that one resource is not being matched to more than one application, thus alleviating the lack of any perfectly up-to-date information about the status of resources, and also helps avoid situa-

tions where the same available resources are assigned to more than one resource petition, thus creating a deadlock in MPI jobs.

The CrossBroker uses a split execution system to enable input/output forwarding on the grid. The InteractiveAgent and InteractiveAgentArguments in the JDL let the user specify the remote agent. On the one hand, when a job is submitted, the CrossBroker is responsible for the staging of the agent to the proper location in the remote nodes and for performing the proper invocation of the agent with the user executable. On the other hand, the agent is in charge of starting the user application (passed as an argument to the agent) and redirect its input and output to the shadow in user machine. We provide with the CrossBroker an agent and shadow (started automatically by CrossBroker in the user machine) based on Condor Bypass[11]. Another agent based on Glogin[12] has also been adapted. In addition, our approach can be easily used to support other agents that are based on other systems similar to Bypass and Glogin.

5 MANAGING MPI JOBS

The CrossBroker provides a reliable submission of parallel jobs on a grid. Once the Scheduling Agent (SA) is notified that an MPI application needs to be executed, the matchmaking process is performed to determine the site (or sites) for executing the application (see Section 5.1). When this is complete, the CrossBroker invokes the Application Launcher (AL) service. This service is responsible for providing a reliable submission of parallel applications. It can handle both intra-cluster MPI applications (compiled either with Open MPI or MPICH-p4) and inter-cluster MPI applications (compiled with PACX-MPI[9] or with MPICH-G2 [10]), depending both on the resources available on the grid and on user-execution needs.

The AL submits a launcher script reserving as many machines as specified by the SA on the selected site(s) using Condor-G[13] job management mechanisms. This launcher script is responsible for obtaining the executable code and the files specified in the InputSandbox attribute of the JDL. After obtaining such files, the AL uses a helper script for starting the parallel application. In the case of the MPI implementations supported, it invokes MPI-START as helper script with the correct configuration values to start the application. MPI-START can be installed on the site or downloaded automatically to the WN if it is not found at runtime.

Support for additional parallel job types (SubJobType *plain* in the JDL) is given with the ApplicationLauncher and ApplicationLauncherArguments attributes of the JDL. The user can specify a launcher script that will be in charge of starting the application in the nodes allocated by the CrossBroker. In addition to the arguments specified in the JDL, the script will also have the user application as argument. Moreover, if the job is interactive, the InteractiveAgent is passed to the launcher script. The script should call this agent when it has completely prepared the environment for the parallel execution of the application.

5.1 MPI Inter-cluster Applications

CrossBroker allows the execution of MPI applications on a grid compiled with PACX-MPI or to be compiled with MPICH-G2, which allows submission to multiple grid sites, thus using the set matchmaking capability of our Resource Searcher for the automatic search of resources. The set matchmaking capability [6] can be guided by the user with the use of the Subjobs attribute in JDL. This attribute specifies the different subjobs that one application is composed of, and for each of them defines a number of nodes to use and any kind of special requirements for that subjob. Figure 5 shows an example of an inter-cluster application requiring 10 CPUs, 5 of them in a cluster with machines with more than 1 GBytes of RAM and 5 of them in a cluster with machines with OpenGL.

```
SubJobs = {
  [ NodeNumber = 5;
   Requirements = Member(OpenGL, other.GlueHostApplicationRTE);]
  [ NodeNumber = 5;
    Requirements = other.GlueHostMainMemoryRAMSize >= 1024; ]
}
```

Fig. 5. SubJobs specification in JDL

Ideally, MPI applications should always run soon after submission. However, there may be situations where not all the remote resources involved in an execution are available, causing the ready resources to stay idle until all the subjobs start. Our job management service features a special mechanism to deal with these situations for batch MPI jobs. As stated in Section 3, whenever a batch MPI application is submitted, instead of the actual application, an agent is submitted to the remote sites. This agent creates two virtual machines for two types of applications: batch MPI on one hand, and sequential, on the other hand. Then, from a logical point of view, MPI batch jobs will run in one virtual machine and sequential jobs will run in the other one. MPI subjobs are submitted to the batch virtual machine and will wait until all the subjobs are ready for execution. Meanwhile, the sequential virtual machine is used to execute other jobs using backfilling scheduling and hence getting a better utilization of resources. In the case of on-line applications, the agent submitted does not create two virtual machines, but starts the application immediately to ensure a faster start-up time.

In order to ensure the co-allocation of the different subjobs that compound one application, the Application launcher follows a two-step commit protocol (detailed in [6]). This protocol ensures the proper co-allocation of the resources and that all the subjobs are started. The Application launcher uses globus APIs to synchronize the MPICH-G2 applications or the PACX-MPI startup-server, depending on the kind of application submitted. Other implementations can be easily supported by the usage of an API provided by the Application Launcher. This API allows the specification of synchronization and submission methods for each MPI implementation.

Our MPI launcher together with Condor-G guarantees co-allocation, error recovery and exactly-once execution semantics for all the MPI jobs supported. Either if the application ends correctly or if there is any problem in the execution of any subjob, the AL records this in a log file that will be checked by the SA, which will take the correct action, in accordance with that information. This provides a reliable once-only execution of the application without user intervention.

6 CONCLUSIONS

Traditionally, many scientific applications submitted to a Grid are executed in a pure batch-mode. Applications run unattended in the background and, consequently, are not able to perform interactive input and output. Adding interactive execution capabilities in grid jobs is one of the main goals of the CrossBroker management system. We have described the main components of the resource management system that we have developed in order to provide automatic and reliable support for interactive and MPI jobs over grid environments.

REFERENCES

[1] Schopt, J.: Ten Actions When Grid Scheduling. In: J. Nabryzke, J. Schopf, and J. Weglarz (Eds.): Grid Resource Management – State of the Art and Future Trends, Kluwer Academic Publishers, 2003.

[2] Bubak, M.—Malawski, M.—Rycerz, K.—Turala, M.: CrossGrid – Tools and Services for Interactive Grid Applications. In: Proc. of the 18th Annual ACM Int. Conf. on Supercomputing, 2004.

[3] The int.eu.grid Project. Available on: `http://www.inteugrid.org`.

[4] Pacini, F.: JDL Attributes Specification. Available on: `https://edms.cern.ch/document/590869`.

[5] Raman, R.—Livny, M.—Solomon, M.: Matchmaking: Distributed Resource Management for High Throughput Computing. In: Proc. of the 7th IEEE Symposium on High Performance Distributed Computing, 1998.

[6] Fernández, E.—Heymann, E.—Senar, M .A.: Supporting Efficient Execution of MPI Applications Across Multiple Sites. In: Proc. of Euro-Par 2006, pp. 383–392.

[7] Foster, I.—Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. In: Intl. J. Supercomputer Applications, Vol. 11, 1997, No. 2, pp. 115–128.

[8] Morajko, A.—Fernández, E.—Fernández, A.—Heymann, E.—Senar, M. A.: Workflow Management in the CrossGrid Project. In: Proc. of EGC 2005, pp. 424–433.

[9] Keller, R.—Gabriel, E.—Krammer, B.—Müller, M.—Resch, M.: Towards Efficient Execution of MPI Applications on the Grid: Porting and Optimization issues. In: J. of Grid Computing, Vol. 1, 2003, No. 2, pp. 133–149.

[10] Karonis, N.—Toonen, B.—Foster, I.: MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. In: J. of Par. and Distrib. Computing (JPDC), Vol. 63, 2003, No. 5, pp. 551–563.

[11] Fernández, E.—Heymann, E.—Senar, M. A.: Resource Management for Interactive Jobs in a Grid Environment. In: Proc. of IEEE Int. Conf. On Cluster Computing (Cluster 2006), Barcelona (Spain), CD-ROM edition, 2006.

[12] Rosmanith, H.—Kranzlmüller, D.: glogin – A Multifunctional, Interactive Tunnel into the Grid. In: Proc. of 5th IEEE/ACM Int. Workshop on Grid Comp., 2004.

[13] Thain, D.—Tannenbaum, T.—Livny, M.: Condor and the Grid. In: Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc., 2003.

[14] Smith, J.—Nair, R.: Virtual Machines: Architectures, Implementations and Applications, Morgan Kaufmann Publishers, 2004.

Enol Fernández graduated in Computer Engineering at the Universidad de La Laguna in 2003. He began the research career during the grade studies in a project related to distributed operating systems. Later on, he moved to the Universitat Autonónoma de Barcelona to join the European CrossGrid Project at the Computer Architecture and Operating Systems Department. He developed the initial version of the CrossBroker Grid Scheduler, a job management system for interactive and parallel jobs. Currently, he is also a member of the int.eu.grid project as main developer of the CrossBroker. In addition, he gives lectures on Computer Architecture. As a result of all this work he will defend his PhD dissertation on 2008.