Managing MPI Applications in Grid Environments*

Elisa Heymann[1], Miquel A. Senar[1], Alvaro Fernández[2], José Salt[2]

[1] Unitat d'Arquitectura d'Ordinadors i Sistemes Operatius
Universitat Autònoma de Barcelona, Barcelona, Spain
{elisa.heymann, miquelangel.senar}@uab.es
[2] Instituto de Física Corpuscular, Valencia , Spain
{alferca, salt}@ific.uv.es

Abstract. One of the goals of the EU CrossGrid project is to provide a basis for supporting the efficient execution of parallel and interactive applications on Grid environments. CrossGrid jobs typically consist of computationally intensive simulations that are often programmed using a parallel programming model and a parallel programming library (MPI). This paper describes the key components that we have included in our resource management system in order to provide effective and reliable execution of parallel applications on a Grid environment. The general architecture of our resource management system is briefly introduced first and we focus afterwards on the description of the main components of our system. We provide support for executing parallel applications written in MPI either in a single cluster or over multiple clusters.

1. Introduction

Grid technologies and concepts started to appear in the mid-1990s. Much progress has been made on the construction of such an infrastructure since then, although some key challenge problems remain to be solved. There are many Grid initiatives that are still working in the prototype arena. And only a few attempts have been made to demonstrate production-level environments up to now. The Compact Muon Solenoid (CMS) Collaboration [1], which is part of several large-scale Grid projects, including GriPhyN [2], PPDG [3] and EU DataGrid [4], is a significant example that has demonstrated the potential value of a Grid-enabled system for Monte Carlo analysis by running a number of large production experiments but not in a continuous way.

Fundamental to any Grid environment is the ability to discover, allocate, monitor and manage the use of resources (which traditionally refer to computers, networks, or storage). The term *resource management* is commonly used to describe all aspects of the process of locating various types of resources, arranging these for use, utilizing

* This work has been partially supported by the European Union through the IST-2001-32243 project "CrossGrid" and partially supported by the *Comisión Interministerial de Ciencia y Tecnología* (CICYT) under contract TIC2001-2592.

them and monitoring their state. In traditional computing systems, resource management is a well-studied problem and there is a significant number of resource managers such as batch schedulers or workflow engines. These resource management systems are designed and operate under the assumption that they have complete control of a resource and thus can implement mechanisms and policies for the effective use of that resource in isolation. Unfortunately, this assumption does not apply to Grid environments, in which resources belong to separately administered domains.

Resource management in a Grid therefore has 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 environment, 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 [5]:

- Resource discovery, which generates a list of potential resources that can be used by a given application. This phase requires the user to have access to a set of resources (i.e. he/she is authorized to use them) and has some mechanism to specify a minimal set of application requirements. These requirements will be used to filter out the resources that do not meet the minimal job requirements.
- Information gathering on those resources and the selection of a best set. In this phase, given a group of possible resources, all of which meet the minimum requirement for the job, a single resource must be selected on which to schedule the job. The resource selection may be carried out by some form of heuristic mechanism that may use additional information about the dynamic state of the resources discovered in the first phase.
- Job execution, which includes file staging and cleanup. Once resources are chosen, the application can be submitted to them. However, due to the lack of standards for job submission, this phase can be made very complicated because it may involve setup of the remote site, staging of files needed by the job, monitoring progress of the application and, once the job is completed, retrieving of output files from the remote site, and removing temporary settings.

The resource management system that we are developing in the CrossGrid project follows the same approach to schedule jobs as described above. However, our system is targeted to a kind of applications that have received very little attention up to now. Most existing systems have focussed on the execution of sequential jobs, the Grid being a large multi-site environment where the jobs run in a batch-like way. The CMS Collaboration constitutes a remarkable example, in which research on job scheduling has also taken into account the location and movement of data, and the coordinated execution of multiple jobs with dependencies between them (when a job X depends on job Y, this means that X can start only when Y has completed).

CrossGrid jobs are computationally intensive applications that are mostly written with the MPI library. Moreover, once the job has been submitted to the Grid and has started its execution on remote resources, the user may want to steer its execution in an interactive way. This is required to analyze intermediate results produced by the application and to react according to them. For instance, in the case where a simulation is not converging, the user may kill the current job and submit a new simulation with a different set of input parameters. From the scheduling point of view, support for parallel and interactive applications introduces the need for some mechanisms that are not needed when jobs are sequential or are submitted in a batch form. Basically, jobs need more than one resource (machine) and they must start *immediately*, i.e. in a period of time very close to the time of submission. Therefore, the scheduler has to search for sets of resources that are all already and wholly available at the time of the job submission. On the other hand, if there are no available resources, some priority and preemption mechanisms might be used to guarantee, for instance, that interactive jobs (which will have the highest priority) will preempt low priority jobs and run in their place.

In this paper, we focus on the description of the basic mechanisms used in our resource management system that are related to the execution of parallel applications on a Grid environment, assuming that free resources are available and no preemption is required. Preemption on grid environments is a complex problem and, to the best of our knowledge, no attempts have been made to address this problem. We are also investigating this issue and have designed certain preliminary mechanisms, which we plan to complete and test them in the near future.

The rest of this paper is organized as follows: Section 2 briefly describes the overall architecture of our resource management services, Section 3 describes the particular services that support submission of MPI applications on a cluster of a single site or on several clusters of multiple sites, and Section 4 summarizes the main conclusions to this work.

2. Overall Architecture of CrossGrid Resource Management

This section briefly describes the global architecture of our scheduling approach. A more detailed explanation can be found in [6]. The scenario that we are targeting consists of a user who has a parallel application and wishes to execute it on grid resources. 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 will take into account the application requirements needed for its execution, as well as certain ranking criteria used to sort the available resources in order of preference.

2. Perform a reliable submission of the application onto the selected resources.
3. Monitor the application execution and report on job termination.

Figure 1 presents the main components that constitute the CrossGrid resource-management services. A user submits a job to a Scheduling Agent (SA) through a web portal. The job is described by a *JobAd* (Job Advertisement) using the EU-Datagrid *Job Description Language* (*JDL*) [7], which has been conveniently extended with additional attributes to reflect the requirements of interactive and parallel applications.

The SA asks the Resource Searcher (RS) for resources to run the application. The main duty of the RS is to perform the matchmaking between job needs and available resources. The RS receives a job description as input, and returns as output a list of possible resources within which to execute the job. The matchmaking process is based on the Condor ClassAd library [8], which has been extended with a set matchmaking capability, as described in [6]. Currently, set matchmaking is used for MPI applications that require a certain number of free CPUs and there is no single cluster that can provide such a number of free CPUs. Set matchmaking generates sets (groups) of clusters so that the overall number of free CPUs in each set fulfils application requirements.

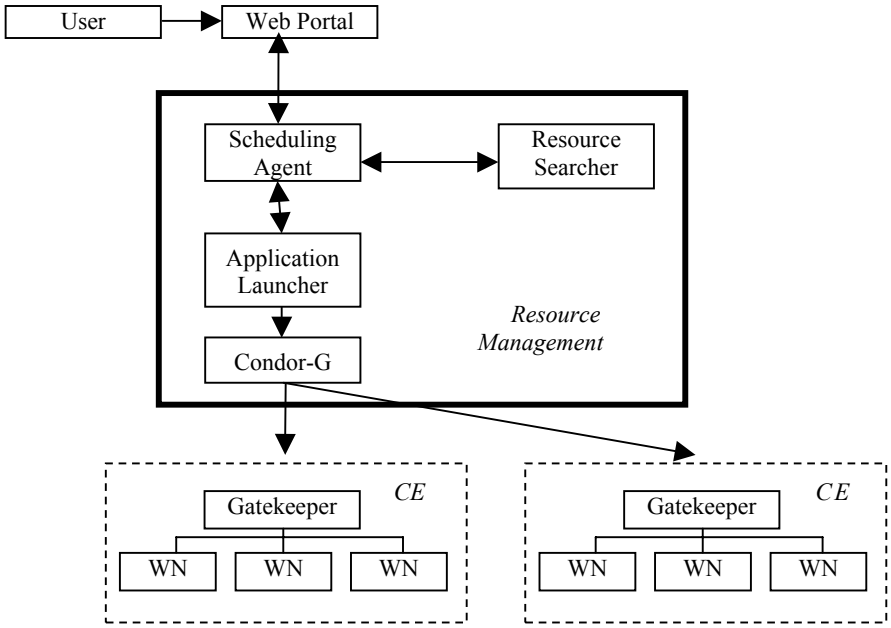

Figure 1. Resource-Management Architecture.

The SA passes the job and the first-selected cluster (or group of CEs), also referred to as Computing Element (CE) in CrossGrid terminology, to the Application

Launcher, who is responsible for the actual submission of the job on the specified CE or groups of CEs.

The Application Launcher is responsible for providing a reliable submission service of parallel applications on the Grid. Currently, two different launchers are used for MPI applications, namely MPICH ch-p4 [9] and MPICH Globus2 [10].

In the following section, both launchers are described.

3. MPI Management

An MPI application to be executed on a grid can be compiled either with MPICH-p4 (ch-p4 device) or with MPICH-G2 (Globus2 device) [13], depending both on the resources available on the grid and on user-execution needs.

On the one hand, MPICH-p4 allows use of machines in a single cluster. In this case, part of the MPICH library must be installed on the executing machines. On the other hand, with MPICH-G2 applications can be submitted to multiple clusters, thus using the set matchmaking capability of the Resource Searcher. However, this approach is limited to clusters where all their machines have public IP addresses. MPICH-G2 applications, unlike MPICH-p4 do not require that the MPICH library is installed on the execution machines.

Taking into account the limitations on IP addresses, the Resource Searcher matches MPICH-p4 applications with single clusters independently of whether they have public or private addresses. The MPICH-G2 application, however, should be matched only with clusters with public IPs. Unfortunately, this information is not announced by the clusters and, therefore, the match generated by the Resource Searcher may include machines with private IPs. As a consequence, the part of the application that is submitted to one of those machines with a private IP will not be started successfully, blocking the whole application. As we explain below, detection of this kind of problem is left to the Application Launcher, who will be in charge of detecting the problem and reacting accordingly.

3.1 MPICH-p4 Management

MPICH-p4 applications will be executed on a single site, as shown in figure 2. Once the Scheduling Agent (SA) is notified that an MPICH-p4 application needs to be executed, the Matchmaking process is performed in order to determine the site for executing the application. When this is complete, the SA launches the application on the selected site following 2 steps:

- Using Condor-G [11] a launcher script is submitted to the selected site (Arrow A in fig. 2). This script is given to the site job scheduler (for

example PBS), which reserves as many machines (worker nodes) as specified in the Condor submission file.

- The script is executed on one such machine, for example in WN1. This script is in charge of obtaining the executable code (Arrow B in fig. 2), as well as the files specified in the *InputSandbox* parameter of the *jdl* file. After obtaining such files, the script performs an *mpirun* call for executing the MPICH-p4 code on the required number of workers.

In this approach, it is assumed that all the worker nodes share the part of the file system where the users are located (traditionally */home*); therefore by transferring the executable file to one worker node, it is accessible to the rest of worker nodes. Additionally it is worth mentioning that *ssh* had been configured to not ask for any password, therefore the MPICH-p4 subjobs can start their execution automatically on the worker nodes.

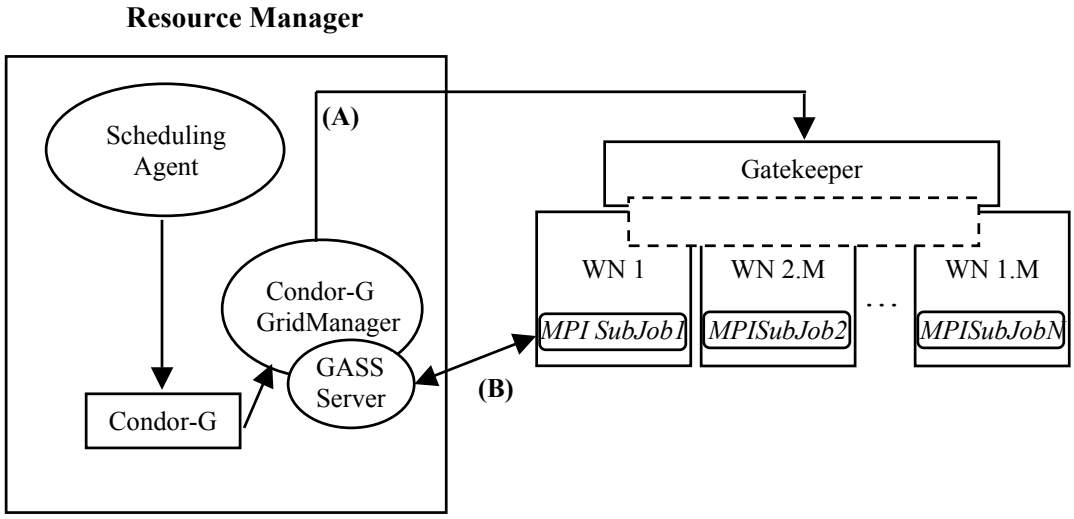

Figure 2. MPI execution on a single site.

3.2 MPICH-G2 Management

When the parallel application needs more machines than the machines provided by any single site, multi-site submission is required. By using MPICH-G2, a parallel application can be executed on machines belonging to different sites.

An MPICH-G2 application can be executed on multiple sites using the *globusrun* command in the following way: `globusrun -s -w -f app.rsl`, the various gatekeepers where the different subjobs of the MPICH-G2 application are expected to

be executed being specified in the *app.rsl* file. The *globusrun* call invokes DUROC[12] for subjob synchronization through a barrier mechanism. But when executing jobs with *globusrun*, the user should be aware of the need to ask for the status of his/her application, resubmitting the application again if something has gone wrong, and so on. In order to free the user of such responsibilities, we propose using Condor-G for a reliable job execution on multiple sites. Our MIPCH-G2 application launcher handles subjob synchronization using the same services provided by DUROC, but also obtains the benefits of using Condor-G.

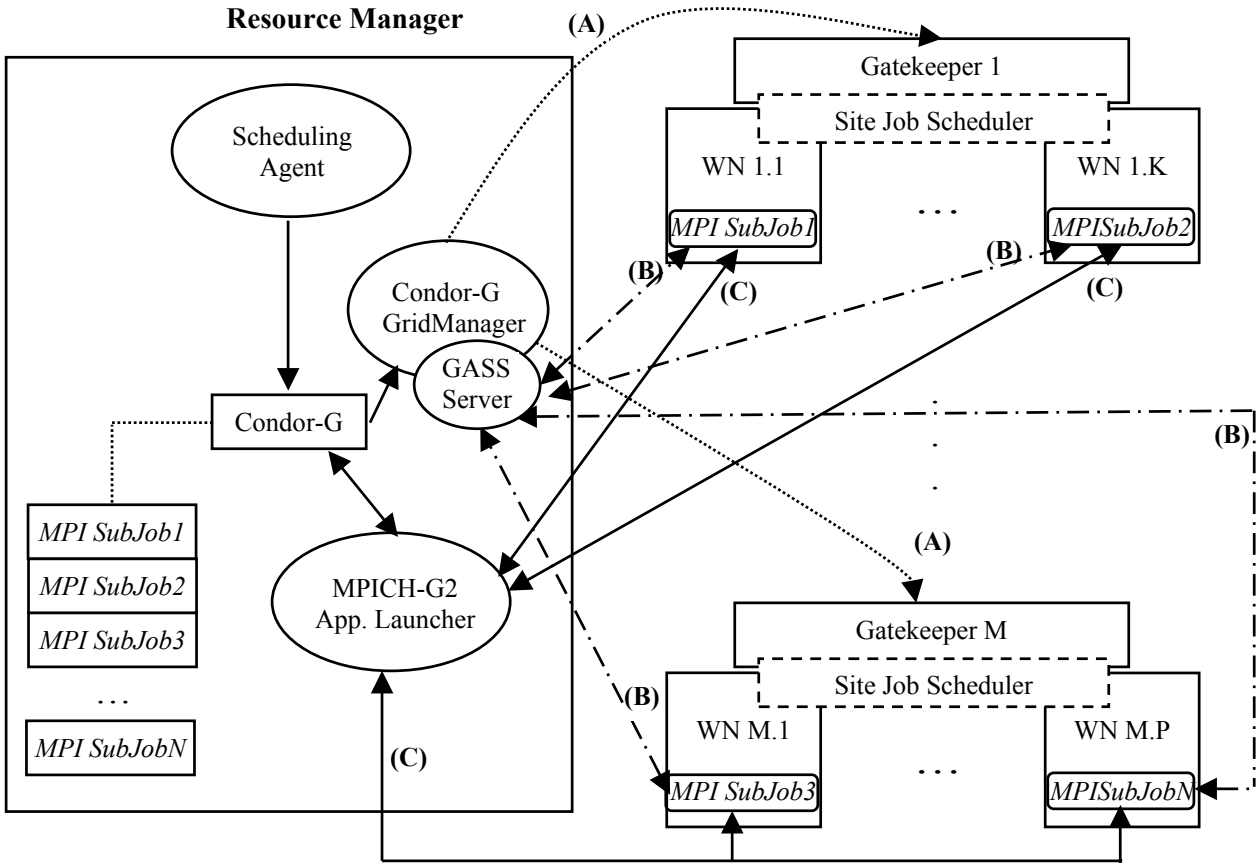

Figure 3. MPI execution on multiple sites

Once the Scheduler Agent (SA) detects that an MPI application is submitted, it launches an MPICH-G2 application launcher (MPI-AL), through Condor-G [11]. This MPI-AL coallocates the different subjobs belonging to the parallel application, following a two-step commit protocol:

- In the first step, all the subjobs are submitted through Condor-G.
- A second step guarantees that all the subjobs have a machine to be executed on, and that they have executed the *MPI_Init* call. This MPICH-G2 call invokes DUROC, and synchronization is achieved by a barrier released by the MPI-AL.

After such synchronization, the subjobs will be allowed to run. Figure 3 depicts how the execution over multiple sites is performed. In this example scenario, we have N subjobs that constitute an MPICH-G2 application. These subjobs will be executed on different sites. For the sake of simplicity, figure 3 only shows 2 sites. The A arrows show this two-steps co-allocation phase. It is important to note that the GASS server is contacted to stage executable files to the remote worker nodes, and to bring the output files back to the submitting machine. This is shown by the B arrows. Once the subjobs are executing on the worker node machines, the MPI-AL monitors their execution and writes an application global log file, providing complete information of the jobs' execution. This monitoring is shown by the C arrows in figure 3, and constitutes the key point for providing reliable execution of the applications and robustness.

If the application ends correctly or if there is any problem in the execution of any subjob, the MPI-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, as we will detail below.

In order to guarantee a reliable execution, the SA monitors all the MPI-ALs submitted; when one of these finishes, it checks the corresponding application global log file written by the finished MPI-AL.

The problems detected can occur at different moments, and can be either temporary or permanent problems. Table 1 shows two problems that can appear before the two-step commit protocol is finished:

Situation detected by the MPI-AL	**SA Action**
A subjob was not executed because a Globus resource was down.	Mark such Globus resource as "unavailable" so it will not be eligible for executing jobs for a certain amount of time. Repeat the Matchmaking process.
A subjob did not start execution but the Globus resource was up.	The same submission will be retried, but if the same situation continues it may be due to a firewall problem in the remote machine. The application will repeat the Matchmaking process.

Table 1. Problems reported by the MPI-AL and handled by the Scheduling Agent.

If the MPI-AL crashes, the SA will submit another MPI-AL that will take over the identification of the crashed item and will control the subjobs of the application.

If the application ends correctly or if there is an abnormal subjob terminarion (program error), the SA will notify the user.

To summarize, the main points as regards the MPI-AL are the following:

- A once-only reliable execution of the application.
- A coordinated execution of the application jobs, which means that the jobs will be executed when all of them have resources to run on.
- A good use of the resources: If a subjob cannot be executed, the whole application will fail, therefore the machines will not be blocked and will be ready to be used by other applications.

4. Conclusions

We have described the main components of the resource management system that we are developing at the EU-CrossGrid in order to provide automatic and reliable support for MPI jobs over grid environments. The system consists of three main components: a Scheduling Agent, a Resource Searcher and an Application Launcher.

The Scheduling Agent is the central element that keeps the queue of jobs submitted by the user and carries out subsequent actions to effectively run the application on the suitable resources. The Resource Searcher has the responsibility of providing groups of machines for any MPI job with both of the following qualities: (1) desirable individual machine characteristics, and (2) desirable characteristics as an aggregate. Finally, the Application Launcher is the module that, in the final stage, is responsible for ensuring a reliable execution of the application on the selected resources. Two different Application Launchers have been implemented to manage the MPICH parallel applications that use the ch-p4 device or the Globus2 device, respectively.

Both launchers take advantage of the basic services provided by Condor-G for sequential applications submitted to a Grid. The launchers also extend these services in order to provide a reliable submission service for MPI applications. As a consequence, our resource management service handles resubmission of failed parallel jobs (due to crashes on the remote resources or failures in the network connecting the resource manager and the remote resources), reliable co-allocation of resources (in the case of MPICH-G2), exactly-once execution (even in the case of a machine crash where the resource manager is running).

Our first prototype has been based on the EU-Datagrid Resource Broker (release 1.4.8). However, this prototype has been mainly used for testing purposes. Our subsequent prototype will be compatible with the next CrossGrid testbed deployment, which is based on EU-Datagrid release 2.0, and will be integrated with two

middleware services (namely, a Web Portal and a Migrating Desktop) providing a user-friendly interface to interact with the Grid.

5. References

[1] K. Holtman. CMS requirements for the Grid. In Proceedings of International Conference on Computing in High Energy and Nuclear Physics (CHEP 2001), 2001.

[2] GriPhyN: The Grid Physics Network. http://www.griphyn.org.

[3] PPDG: Particle Physics Data Grid. http://www.ppdg.net.

[4] European DataGrid Project. http://www.eu-datagrid.org.

[5] Jennifer M. Schopf, "Ten Actions When Grid Scheduling", in Grid Resource Management – State of the Art and Future Trends (Jarek Nabryzki, Jennifer Schopf and Jan Weglarz editors), Kluwer Academic Publishers, 2003.

[6] E. Heymann, M.A.Senar, A. Fernandez, J. Salt, "The Eu-Crossgrid approach for Grid Application Scheduling", to appear in the post-proceedings of the 1st European Across Grids Conference February, LNCS series, 2003.

[7]Fabricio Pazini, JDL Attributes - DataGrid-01-NOT-0101-0_4.pdf, http://www.infn.it/workload-grid/docs/DataGrid-01-NOT-0101-0_4-Note.pdf, December 17, 2001.

[8] Rajesh Raman, Miron Livny and Marvin Solomon, "Matchmaking: Distributed resource management for high throughput computing", in Proc. Of the seventh IEEE Int. Symp. On High Perfromance Distributed Computing (HPDC7), Chicago, IL, July, 1998.

[9] W. Gropp and E. Lusk and N. Doss and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard", Parallel Computing, 22(6), pages 789-828, 1996.

[10] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Gridenabled implementation of the message passing interface. *Journal of Parallel and Distributed Computing*, *to appear,* 2003.

[11] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "Condor-G: A Computation Management Agent for Multi-Institutional Grids", Journal of Cluster Computing, vol. 5, pages 237-246, 2002.

[12] K. Czajkowsi, I. Foster, C. Kessekman. "Co-allocation services for computational Grids". Proceedings of the Eighth IEEE Symposium on High Performance Distributed Computing, IEEE Computer Society Press, Silver Spring MD, 1999.

[13] N. Karonis, B. Toonen, I. Foster. "MPICH-G2: A Grid-enabled implementation of the Message Passing Interface". Journal of Parallel and Distributed Computing, 62, pp. 551-563, 2003.