Reliable Scheduling of MPI Applications on a Grid Environment[1]

Enol Fernández[2], Elisa Heymann[2], Miquel A. Senar[2], Emilio Luque[2], Álvaro Fernández[3]

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.**

Keywords—**Grid Computing, MPI, Scheduling.**

I. INTRODUCTION

Grid technologies 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 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.
- Information gathering on those resources and the selection of a best set.
- Job execution, which includes file staging and cleanup.

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. 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.

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.

[1] This work has been supported by the MCyT-Spain under contracts TIC 2001-2592, TIC2002-10430-E, TIC2002-12000-E, the European Union through the IST-2001-32243 project "CrossGrid" and partially supported by the Generalitat de Catalunya- Grup de Recerca Consolidat 2001SGR-00218.
[2] Unitat d'Arquitectura d'Ordinadors i Sistemes Operatius, Universitat Autònoma de Barcelona, Barcelona, Spain.
{elisa.heymann, miquelangel.senar, enol.fernandez, emilio.luque}@uab.es
[3] Instituto de Física Corpuscular, Valencia , Spain. alferca@ific.uv.es.

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 that we're currently dealing with.

The rest of this paper is organized as follows: Section II briefly describes the overall architecture of our resource management services, Section III 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 IV summarizes the main conclusions to this work.

II. General Architecture of CrossGrid Resource Management

This section briefly describes the global architecture of our scheduling approach. 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:

- Select the "best" resources that a submitted job 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.
- Perform the necessary steps to guarantee the effective submission of the job onto the selected resources. The application is allowed to run to completion.
- 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 *User Interface* (UI)machine. This is a machine connected to the Resource Broker, and it is in the UI where the user level basic commands are executed. The job is described by a *JobAd* (Job Advertisement) using the EU-Datagrid *Job Description Language* (*JDL*) [6], which has been conveniently extended with additional attributes to reflect the requirements of interactive and parallel applications, which are explained in section III-A.

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 [7], which has been extended with a set matchmaking capability, as described in [8]. 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.

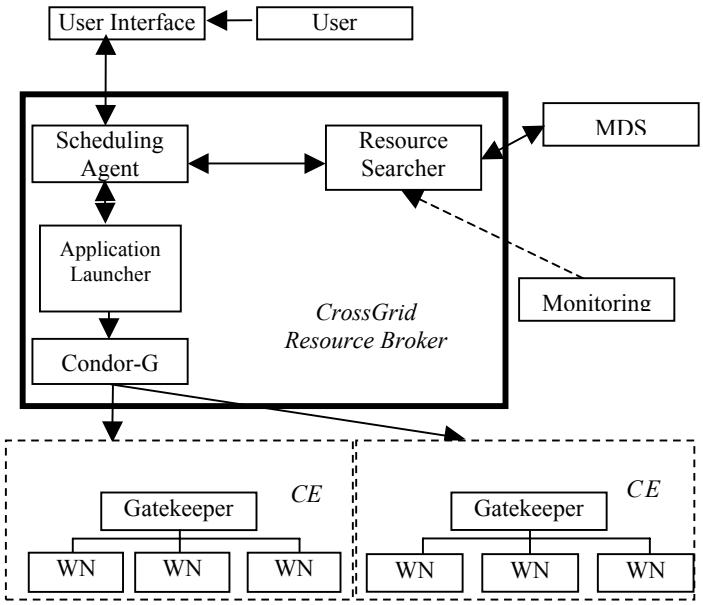

Fig. 1. Resource-Management Architecture.

Computing resources in the CrossGrid architecture are available as Computing Elements (CE), which provide the abstraction of a local farm of Working Nodes (WN). This local farm (or CE) is accessed through a Gatekeeper. Thus, the list of resources returned by the Resource Searcher consists of a Computing Elements list, which may eventually be grouped into sets, if the submitted job is an MPICH-G2 job and its requirements cannot be fulfilled by a single CE alone.

Subsequently, the Scheduling Agent selects a CE or a group of CEs on which to run the MPI job, according to the following criteria:

1. Groups of CEs with fewer numbers of different CEs will be selected first. This criterion tends to run MPI on a single cluster, which will avoid large message latencies between tasks allocated in different clusters.

2. If there is more than one group with the same number of CEs, the group having best global rank will be selected first. Ranks are assigned to each CE (or groups of CEs) according to certain performance metrics (e.g. overall MFLOPS, main memory, etc.).

The SA passes the job and the first-selected CE (or group of CEs) to the Application Launcher, who is responsible for the actual submission of the job on the specified CE. Due to the dynamic nature of the Grid, the job submission may fail on that particular CE. Therefore, the Scheduling Agent will try the other CEs from the list returned by the Resource Searcher until the job submission succeeds or fails. In the later case, the Scheduling Agent notifies the user of the failure.

The SA will keep permanent information about all user jobs and it will ensure that all necessary resources are co-allocated before passing a parallel job to Condor_G [9]. 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 [10] and MPICH Globus2 [11].

III. Resource Management Components

We now describe certain details on the main components introduced in the previous section, namely, the Resource Searcher and the Application Launcher.

A. Resource Searcher

The main duty of the Resource Searcher is to perform the matchmaking between job needs and available resources. The RS receives a job description as input, and produces as output a list of possible resources in which to execute the job. Resources are grouped into sets. Each set is a combination of Computing Elements that provide the minimum amount of free resources, as specified by the *JobAd*. As we said before, JobAds are described using the Job Description Language (JDL) adopted in the EU-Datagrid project. This language is based on the Condor ClassAd library.

The matchmaking process carried out by the Resource Searcher is also implemented with the Condor ClassAd library. With this library, jobs and resources are expressed as ClassAds; two of these match if each of the ClassAd attributes evaluate to true in the context of the other ClassAd. Because the ClassAd language and the ClassAd matchmaker were designed for selecting a single machine on which to run a job, we have added several extensions to be applied when a job requires multiple resources (i.e. multiple CEs, in CrossGrid terminology).

With these extensions, a successful match is defined as occurring between a single ClassAd (the *JobAd*) and a ClassAd set (a group of CEs ClassAds). Firstly, the *JobAd* is used to place constraints on the collective properties of an entire group of CEs ClassAd (e.g., the total number of free CPUs has to be greater than the minimum number of CPUs required by the job). Secondly, other attributes of the *JobAd* are used to place constraints on the individual properties of each CE ClassAd (e.g., the OS version of each CE has to be Linux 2.4).

The selection of resources is carried out according to the following steps:

- First step: a list is obtained of single CEs that fulfill all job requirements referring only to required individual characteristics. Currently, these are the requirements that are specified in the *Requirements* section of the file describing the job using JDL. This step constitutes a pre-selection phase that generates a reduced set of resources suitable for executing the job request in terms of several characteristics such as processor architecture, OS, etc.
- Second step: from the list mentioned above, groups of CEs are made to fulfill collective requirements. For example, an attempt is made to fulfill the total number of CPUs required by a job by "aggregating" individual CEs. In the case of the number of CPUs required by the job, for instance, the Resource Searcher aggregates CEs to guarantee that the total number of free CPUs in the groups of CEs is larger than `NumCPU`, as described in the *JobAd*.

Our current search procedure is not exhaustive, as it does not compute the power set of all CEs. This means that, in an example such as the one shown in figure 2a, for four suitable CEs {CE1, CE2, CE3 and CE5}, only two solutions are provided: {CE2}, {CE1, CE3} {CE1, CE5}. Other possible solutions, such as {CE1, CE3,CE5}, are not considered because one subset of the CEs has already been included in a previous group.

As an example of the functioning of the resource selection we now show the results obtained when executing the '*edg-job-list-match*' command to get the list of resources on which to execute the parallel MPI job described in an *.jdl* file. When this commands is issued, the RB contacts the MDS to get the information about available CEs.

The *jdl* file shall contain the specifications and requirements of the job, and also the new fields that we have established to correctly define the MPI jobs. The interesting fields related to test this module are:

JobType: Field that defines that is a MPI job. Possible values are: "normal" - (default) common sequential job.
"mpich" - defines an MPI job compiled with the ch_p4 device.
"mpich-g2" - defines an MPI job compiled with the G2 device.

NodeNumber Field that defines the required number of cpus to execute the MPI job.

Fig. 2 depicts an *jdl* example file that looks for groups of CEs whose queue type is PBS and that have at least 10 free CPUs in the group, to run the MPICH G2 job named *mpi_app*.

```
Executable = "mpi_app";
JobType = "mpich-g2";
NodeNumber = 10;
Arguments = "-n";
StdOutput = "std.out";
StdError = "std.err";
Requirements = other.GlueCEInfoLRMSType=="pbs";
Rank = other.GlueHostBenchmarkSI00;
OutputSandbox= {"std.out","std.err"};
```

Fig 2. JDL example file

Such *jdl* file is submitted to RB that supports this new *jdl* syntax. The command used to get the available CEs is: **edg-job-list-match file.jdl**

It has to be noted the utilization of the Glue Schema in the Rank and Requirement attributes. The StdOutput and StdError fields specify the files where the standard output and error will be redirected. The output of the command contains a list of resources or groups of resources where to execute the parallel job (see Figure 3). Such list is composed of:

- Groups of elements that contain only 1 CE, so the job could be submitted to just one CE or cluster. This is the best desirable situation. For example, the the CE **ce001.grid.ucy.ac.cy:2119** has all 10 free CPUs and a global rank (based on the SI00 of that CE) of 650. This resource will be the first selected by the Application Scheduler.

```
Connecting to host cg07.ific.uv.es, port
7772
*****************************************
            GROUPS OF CE IDs LIST
 The following groups of CE(s) matching your
job requirements have been found:

*Groups with 1 CEs*   *TotalCPUs* *FreeCPUs*

[Rank=650]
 ce001.grid.ucy.ac.cy:2119    10         10
[Rank=630]
 cluster.ui.sav.sk:2119       16         16
[Rank=400]
 zeus24.cyf-kr.edu.pl:2119    58         57

*Groups with 2 CEs*   *TotalCPUs* *FreeCPUs*

[Rank=440 TotalCPUs=12 FreeCPUs=12]
 cagnode45.cs.tcd.ie:2119      4          4
 ce100.fzk.de:2119             8          8
[Rank=498 TotalCPUs=10 FreeCPUs=10]
 ce01.lip.pt:2119              2          2
 ce100.fzk.de:2119             8          8

 *Groups with 4 CEs*  *TotalCPUs* *FreeCPUs*

[Rank=435.6 TotalCPUs=10 FreeCPUs=10]
 cagnode45.cs.tcd.ie:2119      4          4
 ce01.lip.pt:2119              2          2
 cg01.ific.uv.es:2119          2          2
 cgnode00.di.uoa.gr:2119       2          2
```

Fig 3. Example of MPI application submission over CrossGrid sites and the result obtained by the Resource Selector.

- After single CEs, groups of CEs that fulfill the requirements are formed. In this case we find 2 groups with 2 CEs suitable for executing our job. The best one, according with the rank is the group formed by **`cagnode45.cs.tcd.ie`** and **`ce100.fzk.de.`** The computed rank of 440 is not the average of the ranks of the components (400 and 460, that would be 430) but the weighted rank calculated considering the number of free cpus of each component.
- As it can be seen there are no possible groups with 3 CEs that group the required number of cpus that we are asking for, and that are not taked into account in the previous groups (with 1, or 2 CEs). However we can find 4 groups with 4 CEs.

CEs are sorted according to a Rank expression provided by the user in the JobAd. According to the Rank expression (e.g., Average Spec Int benchmark in our example, as an indication of the computational power), the Resource Searcher sorts the suitable CEs in descending order. This means that the most desirable CEs or groups of CEs will be first. It is worth noting that selection of multiple resources has also been applied in [12]. In contrast to our approach, only the best resource or group of resources is selected. In our work, several choices are generated so that the final decision relies on the Scheduling Agent, which is able to try alternatives in the case of failing to actually submit the job in a given group of CEs.

The Resource Searcher currently supports information collection from the Globus Metacomputing Directory Service (MDS). It is also planned to add support for other resource-information systems, such as R-GMA, from the EU-DataGrid project, which will be also used to collect information obtained by different monitoring tools currently under development in the CrossGrid project.

B. Application Launcher

This service is responsible for providing a reliable submission service of parallel applications on the Grid. It spawns the job on the given machines using Condor-G [7] job management mechanisms. 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), 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. MPICH-G2 applications, unlike MPICH-p4 do not require that the MPICH library is installed on the execution machines. The MPICH-G2 launcher coordinates the start-up of all MPI tasks in each remote cluster.

An MPI job submitted to the Grid may fail due to many different reasons, such as firewalls, machines with private IP addresses, reboot of machines, etc.). Therefore it is fundamental to provide fault tolerant services. Our MPICH-G2 launcher together with Condor-G guarantees co-allocation, error recovery and exactly-once execution semantics for MPICH-G2 jobs and constitutes a reliable submission service that substitutes the submission services provided by the Globus toolkit.

1) MPICH-p4 Management

MPICH-p4 applications will be executed on a single site, as shown in figure 4. 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 a launcher script is submitted to the selected site (Arrow A in fig. 4). 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. 4), 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.

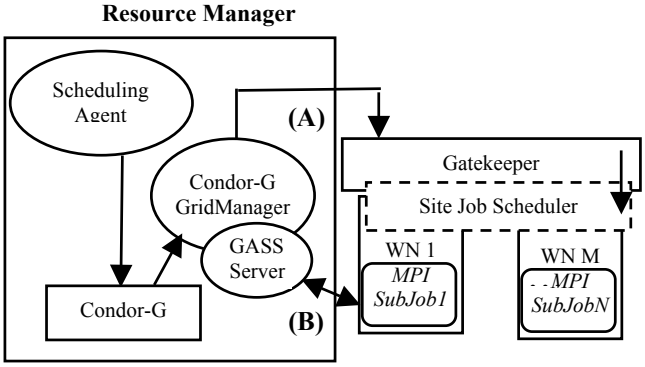

Fig 4. MPI execution on a single site.

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[13] 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. The main benefits offered by the MPICH-G2 Application Launcher are the following:

- An once-only execution of the application.
- A coordinated execution of the application subjobs, which means that the subjobs will be executed when all of them have resources to run on.
- A reliable 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.

Once the Scheduler Agent (SA) detects that an MPI application is submitted, it launches an MPICH-G2 application launcher (MPI-AL), through Condor-G. Figure 5 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 5 only shows 2 sites. 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. The A arrows in fig 5 show the subjobs submission to the remote machines. 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 in fig 5.
- 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. Once the subjobs are executing, 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 5, and constitutes the key point for providing reliable execution of the applications and robustness. Either 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.

Table 1 shows the problems that can appear and the corresponding actions taken. By handling adequately all these problems a reliable MPI execution is guaranteed.

TABLE 1. PROBLEMS REPORTED BY THE MPI-AL AND HANDLED BY THE SCHEDULING AGENT.

Problem Detected	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.
The two-step commit protocol cannot be completed within a limited amount of time.	All the subjobs will be killed, and then the same submission will be retried. If the submission continues failing on the same set of machines, the Matchmaking process will be repeated.
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 subjobs are after the two-steps commit protocol, they will continue their execution without noticing the MPI-AL replacement, otherwise they will be killed and the whole submission will be repeated.
Abnormal subjob termination.	The SA will notify the user.

The user submits jobs from the UI using the command line tools and *jdl* files which describe the job. The jdl file must specify the mpich-g2 jobtype and the number of nodes needed to execute the application as the one depicted in Figure 2. The command used to submit such file is: *edg-job-submit file.jdl*

The output of the command (see Figure 6) indicates that the job has been sent to the RB and now the user must check its status using the edg-job-status command. The job will pass through three different states: Waiting, Running and Done.

When the job has finished, the user can get all the output generated by job using the command *edg-job-get-output*.

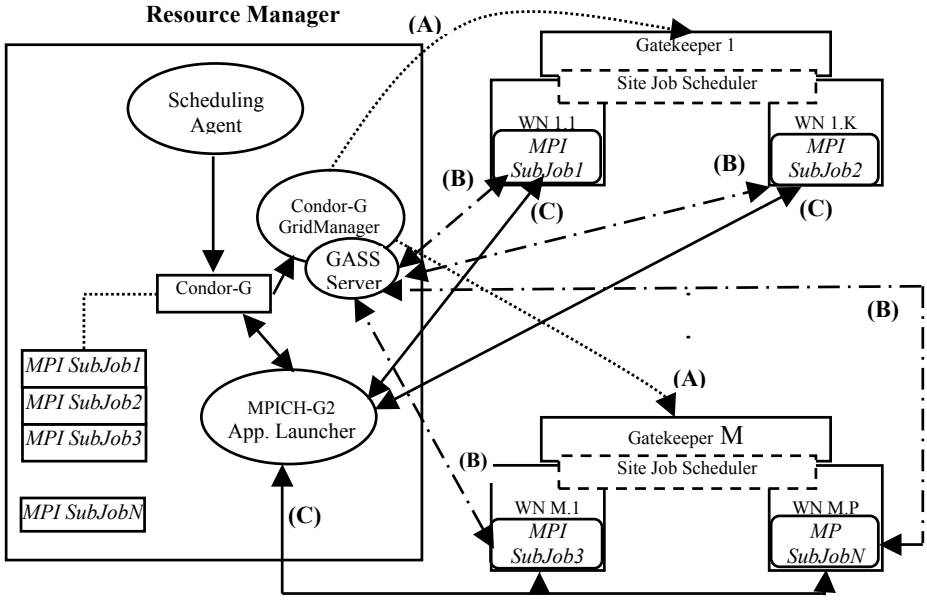

Fig 5. MPI execution on multiple sites

```
Connecting to host aow5grid.uab.es, port 7772
Logging to host aow5grid.uab.es, port 9002

**********************************************************
                    JOB SUBMIT OUTCOME
The job has been successfully submitted to the Network Server.
Use edg-job-status command to check job current status. Your job
identifier (edg_jobId) is:

- https://aow5grid.uab.es:9000/jR0hjTzOlyFkRkpP_i1R8Q
**********************************************************
```

Fig 6. Sample of job submission command.

IV. 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 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.

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 current prototype is part of the CrossGrid release 1.0 and has been deployed at FZK (Germany) at beginning of June, being the central job manager for the whole CrossGrid production testbed since then.

V. 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] 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.
[7] 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.
[8] E. Heymann, M.A.Senar, A. Fernandez, J. Salt, "The Eu-Crossgrid approach for Grid Application Scheduling", Proc. of the 1st European Across Grids Conference, LNCS series vol. 2970, 2003.
[9] 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.
[10] 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.
[11] 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.
[12] Chuang Liu, Lingyun Yang, Ian Foster, Dave Angulo, "Design and Evaluation of a Resource Selection Framework for Grid *Applications*, *Proceedings of IEEE International Symposium on High Performance Distributed Computing (HPDC-11),* Edinburgh, Scotland, July, 2002.
[13] 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.