Supporting Efficient Execution of MPI Applications across Multiple Sites *

Enol Fernández, Elisa Heymann, and Miquel Àngel Senar

Departament d'Arquitectura de Computadors i Sistemes Operatius
Universitat Autònoma de Barcelona, Barcelona, Spain
enol@aomail.uab.es, {elisa.heymann, miquelangel.senar}@uab.es

Abstract. One of the main goals of the CrossGrid Project [1] is to provide explicit support to parallel and interactive compute- and data-intensive applications. The CrossBroker job manager provides services as part of the CrossGrid middleware and allows execution of parallel MPI applications on Grid resources in a transparent and automatic way. This document describes the design and implementation of the key components responsible for an efficient and reliable execution of MPI jobs splitted over multiple Grid sites, executed either in an on-line or batch manner. We also provide details on the overheads introduced by our system, as well as an experimental study showing that our system is well-suited for embarrassingly parallel applications.

1 Introduction

Large-scale Grid computing requires job-management services that address new concerns arising in Grid environments. 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 [2]:

- Resource discovery, which generates a list of potential resources to be used.
- Information gathering on those resources and the selection of a best set.
- Job execution, which includes file staging and cleanup.

* This work was made in the frame of the "int-eu.grid" project (sponsored by the European Union), and supported by the MEyC-Spain under contract TIN 2004-03388, and partially supported by the NATO under contract EST.EAP.CLG 981032.

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 in the CrossGrid project follows the same approach in scheduling jobs. However, our system, known as CrossBroker, is targeted to the kinds of 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. Crossgrid jobs are computationally intensive applications mostly written with the MPICH library using the Globus2 device [3], taking advantage of being executed on multiple Grid sites.

From the scheduling point of view, support for parallel applications introduces the need for co-allocation. There are studies [4][5] that evaluate different co-allocation strategies, although the kind of jobs and grid environment these use are not applicable in CrossGrid and are focused on simulation.

To the best of our knowledge, only a basic support for running MPICH-G2 jobs is included in the Globus Toolkit by using the *globusrun* command and the DUROC services [6]. However this command requires a manual intervention of the user to discover and select resources, and to stage all necessary files to the remote sites and it does not support a reliable co-allocation mechanism to synchronize the start-up of all subjobs. GCM [7] deals with the execution of multi-site jobs using PACX-MPI [8], but it does not include a mechanism for the automatic selection of sites. Our job-management service supports MPICH-G2 job execution by performing the three main scheduling phases in an automatic and reliable way.

The rest of this paper is organized as follows: Section 2 briefly outlines the overall architecture of our resource-management services, Section 3 describes the particular services that support submission of MPI applications on a Grid environment. Section 4 describes some experimental evaluation of our system, and Section 5 summarizes the main conclusions to this work.

2 General Architecture of the CrossBroker

This section briefly describes the global architecture of our scheduling approach. A more detailed explanation can be found in [9]. The scenario that we are targeting consists of a user who has a parallel application and wishes to execute this on grid resources. The user can submit the job in either an on-line or batch manner. On-line submission is made when the application must start *immediately*, i.e. in a period of time very close to the time of submission. This kind of submission is suitable for interactive applications. It is worth observing that batch submissions do not require an immediate application start.

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 exe-

cution. The most important requirement for on-line jobs is the availability of free machines at submission time; therefore, if there are no free machines, the job will be cancelled. In the case of batch submission, the application can wait for a free slot in the Grid sites and also for resources where other specified requirements are satisfied.
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. 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) [10], which has been conveniently extended with additional attributes to reflect the requirements of parallel applications.

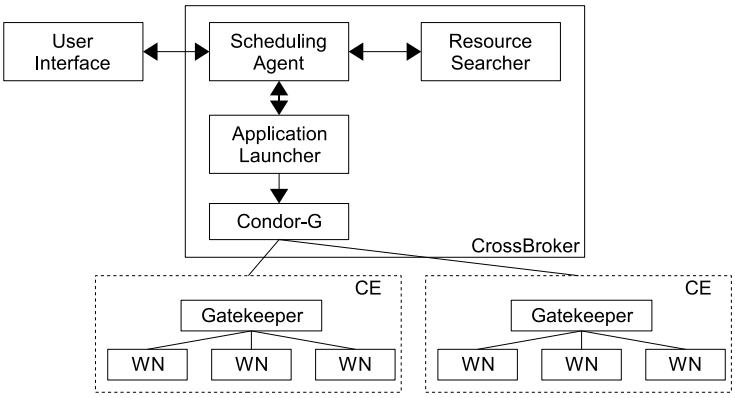

Fig. 1. CrossBroker Resource-Manager Architecture.

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 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. The matchmaking process is based on the Condor ClassAd library [11], which has been extended with a set matchmaking capability, as described in [9].

The SA then selects the best resource (or group of resources) from the list returned by the RS taking into account its current state and the job requirements. The computing resources (or group of resources), also referred to as Computing Element (CE) in CrossGrid terminology, are passed to the Application Launcher,

which is responsible for the co-allocation and 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. Currently, two different launchers are used for MPI applications, one allowing execution on one site, described in detail in [9], and one allowing execution on multiple sites, described in the following section.

3 MPICH-G2 Job Management

An MPI application for grid execution has to be compiled with MPICH-G2 [3], a device which allows the submission to multiple grid sites, thus using the set matchmaking capability of our Resource Searcher for the automatic search of resources.

As we have already mentioned, an MPICH-G2 application can be executed on multiple sites using the *globusrun* command. The *globusrun* call performs subjob synchronization through a barrier mechanism. But when executing jobs with *globusrun*, it falls to the users to decide which sites to use, and it is these same users who should be aware of the need to ask for the status of their own application, resubmitting the application again if something is amiss, and so on. Any failure or delay in the startup of a subjob may block permanently the application given that the remaining subjobs will stay within the synchronization barrier. As a consequence, resources will be occupied but no progress will be achieved in application execution.

The lack of reliability exhibited by the *globusrun* command has been overcome by Condor-G [12], which constitutes a dependable submission system for the Grid. Unfortunately, Condor-G only supports sequential applications. We have modified the submission of MPICH-G2 jobs in such a way that the whole application is decomposed into a set of independent tasks - submitted to the Grid - which are submitted to Condor-G (and treated as sequential tasks). Additionally, we have included the necessary synchronization actions within each task so as to generate a reliable co-allocation of all tasks. We can thus react to the synchronization-related problems experienced by *globusrun* and avoid any blocking situation during the launching phase of the job.

Ideally, MPI applications should always run soon after submission. However, there may be situations in which not all the remote resources involved in an execution are available, causing the ready resources to stay idle until all subjobs start. Our job-management service features a special mechanism to deal with these situations for batch MPI jobs. Whenever a batch MPI-G2 application is submitted, an agent (rather than the actual application) is submitted to the remote sites. This agent, based on Condor Glide-In [12], is used to gain control of remote machines independently of the local-site job scheduler. Each machine acquired by the agent, is configured as two virtual machines, in order to create a separate group of dedicated resources for two types of applications: batch MPI

on the one hand, and sequential, on the other hand. From a logical point of view, MPI batch jobs will then run on one virtual machine and sequential jobs will run on the other one. MPICH-G2 subjobs are submitted to the batch virtual machine and will wait until all subjobs are ready for execution. Meanwhile, the sequential virtual machine is used to execute other jobs using backfilling scheduling, hence attaining better utilization of resources. In the case of on-line applications, the agent submitted does not create two virtual machines, but rather immediately starts the application to ensure a faster start-up time.

In order to ensure the co-allocation of the different subjobs that make up one application, the Scheduler Agent launches an MPICH-G2 application launcher (MPI-AL), through Condor-G. This MPI-AL follows a two-step commit protocol:

- In the first step, all the subjobs (with their agents) are submitted to the remote sites.
- A second step guarantees that all subjobs have a machine for their execution, and that they have executed the MPI Init call. Synchronization is achieved through a barrier released by the MPI-AL. After such synchronization, the subjobs will then be allowed to run.

In order to avoid blocking situations, the MPI-AL will wait for several minutes for on-line jobs to execute their MPI Init call. If this call is not performed before the time is exhausted, the whole job will then be aborted. In the case of batch jobs, time-out will occur when the site's local scheduler removes the job.

Figure 2 depicts how execution over the multiple sites of a batch job is performed. In this example scenario, we have N subjobs constituting an MPICH-G2 application. These subjobs will be executed on different sites. For the sake of simplicity, Fig. 2 only shows 2 sites. The A arrows show subjobs submission to the remote machines. These subjobs will stage agent executable and will start it to gain control of the node. Once the virtual machines are available, the actual application is submitted to the batch virtual machine. This is shown by the B arrows. Once the subjobs are executing on the remote machines, the MPI-AL releases the barrier and starts monitoring 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 Fig. 2, and constitutes the key point for providing both reliable application execution and robustness.

In the event of the application ending correctly or of there being a problem with the execution of any subjob, the MPI-AL records this in a log file that will be checked by the SA, which will then take the correct action, in accordance with that information. This provides a reliable once-only execution of the application without user intervention.

4 Experimental Results

In this section we present an experimental evaluation of our system. First, we measure the overhead introduced by our software when running MPICH-G2 jobs and following this, we evaluate the performance of a real application executed on the Grid.

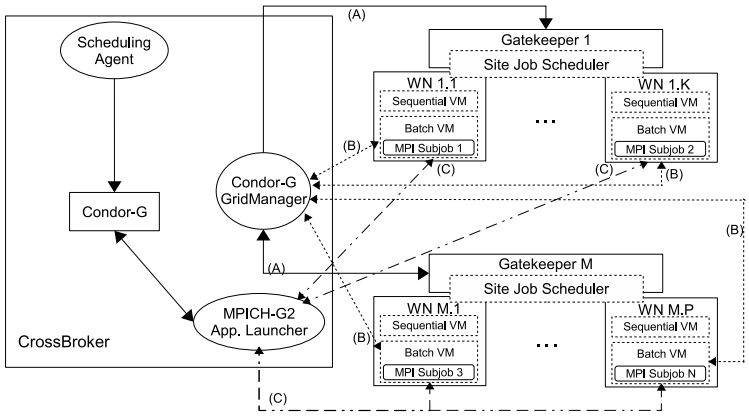

Fig. 2. MPI execution on multiple sites

4.1 MPICH-G2 Overhead

Submitting jobs to the Grid by using the CrossBroker incurs an initial overhead due to the different actions taken before the real job execution. This overhead depends on whether the MPICH-G2 job will run in an on-line or batch manner. Maximum overhead is incurred in the second case because it requires the following steps:

1. Submission of the different subjobs to the remote sites. This involves contacting the Globus gatekeeper, which in turns contacts the site job scheduler to create a basic job that starts our agent in a particular node.
2. Once the basic job has started in the remote site, the agent files are downloaded from the CrossBroker to the node that will execute the application.
3. Virtual Machine set-up: the virtual machines are created. The CrossBroker is notified, which in turn will submit the user application to the agent.
4. Subjob start-up: user application files are downloaded and the job is started in the remote node.

We have submitted a synthetic MPICH-G2 application in order to measure the impact of the different steps. The sites used were the following:

- UAB: cluster in Barcelona with 6 heterogeneous CPUs. The CrossBroker used for these tests is also located in Barcelona.
- FZK: remote cluster with 16 CPUs (4 nodes with 4 CPUs each) located in Karlsruhe (Germany).
- IFCA: remote cluster with 6 CPUs in 2 dual nodes. This cluster is located in Santander (Spain).

Figure 3 shows the time (in seconds) from job submission until the application starts running using different combinations of the above sites. In addition to the

one-site submission, submissions using CPUs from two and three sites are also shown. When more than one site was used, CPUs have were distributed equally among all sites. The time obtained is the sum of the four steps mentioned above. The figure shows that, in general, overhead mostly depends on the sites used. With the increase in the number of sites involved in an execution, this overhead also increases. It should be observed that these measures have been obtained for a worst-case scenario, in which all the steps would be taken.

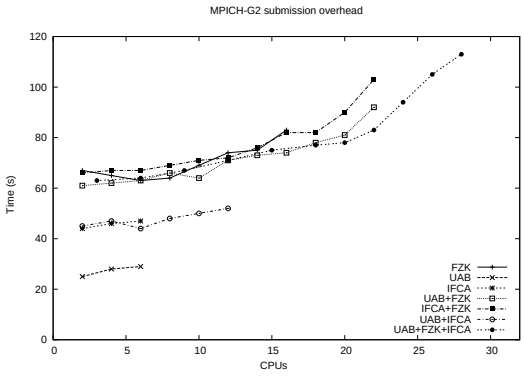

Fig. 3. MPICH-G2 submission overhead for multiple sites.

In order to show the real influence of the different steps involved in application submission, Fig. 4 depicts the first step (submission to the remote sites) and agent download (second step) for the same scenario. Globus submission depends on the queue status of the site's local scheduler. In these tests, submission was made to sites with empty local queues (PBS, Condor), hence the subjobs start as soon as possible. As can be seen in Fig. 4, site submission remains almost constant, despite the number of CPUs used, and depends on the sites used: UAB and FZK jobs start earlier than those at IFCA. This is the minimum delay for application execution in our environment, and is similar to that obtained using *globusrun* directly or when using the on-line scheduling in CrossBroker (the second and third step ares not executed in such a case).

Figure 4 also shows how agent download is the most time-consuming step taken in submitting MPICH-G2 applications. This time depends on the limited bandwidth between the CrossBroker machine and each of the nodes in which the jobs is to be executed. As the number of CPUs increases, bandwidth is shared among all these nodes and the downloading process takes longer. The download takes longer in sites located at greater distance (FZK), i.e. having less bandwidth between the site and the CrossBroker. Downloading time is limited to less than a minute, which is negligible for batch applications intended to run for a much longer period of time.

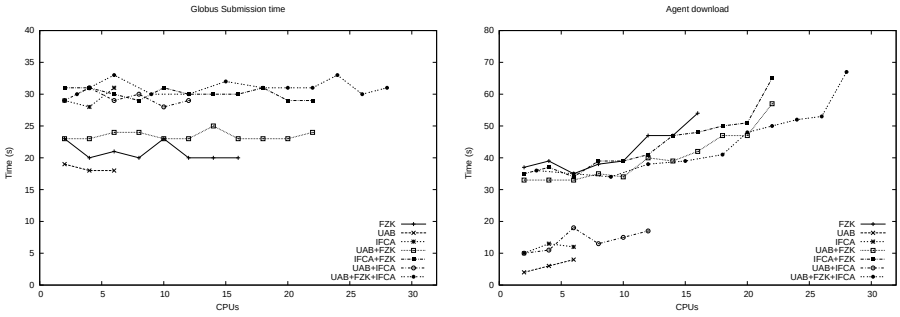

Fig. 4. Left: Globus submission time. Right: Agent download time.

The third step, virtual machine set up, is the time elapsed from the agent file download to the creation of the virtual machines on the remote machines. This time is usually around one second, immediately after the CrossBroker submits the actual application to the batch virtual machine. The last step involves downloading the application binaries. As in the case of the agent, the downloading process depends on the sites involved. This step can be avoided if the user specifies a pre-staged binary in the job description or else makes use of the storage facilities available in the remote sites.

Although the overall overhead is not significant for batch applications, download times for the agent could be avoided by permanently installing the needed files in the remote nodes. In such a case, the CrossBroker would therefore only need to submit a simple job that initiates the agent without any previous download.

4.2 MPICH-G2 Application Execution

MPICH-G2 allows the execution of any MPI application using different-cluster nodes. Applications making heavy use of collective operations that are fairly sensitive to high-latency links are not suitable for this kind of environments. However, there are many applications that exhibit a computation/communication ratio that make them attractive for executiong over multiple sites. Many embarrassingly parallel applications are suited to such applications.

As an example, we have used a Master-Worker application developed in the CrossGrid Project to measure the impact of multiple-site execution. This application trains a neural network to find Higgs Boson [13]. The master node assigns a list of files with input data to each of the workers, and the training is repeated until the obtained error reaches a certain bound. The needed files are downloaded from each of the workers using replica management tools [14].

We have executed the application on the same sites used in section 4.1. In Fig. 5 the execution time is shown for the same CPU combination shown in the previous subsection. Depicted time (in seconds) includes the overhead for on-line

scheduling, so implying there is therefore no creation of virtual machines in the remote nodes. This overhead is around 20 to 30 seconds, depending on the used sites.

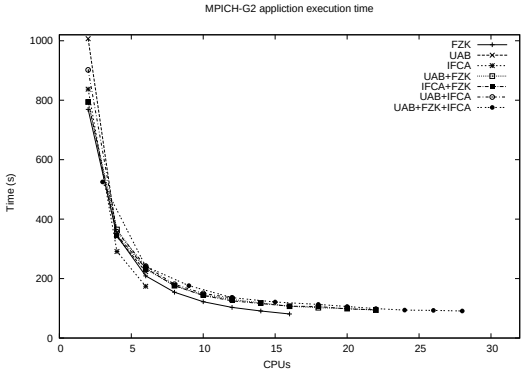

Fig. 5. Neural-net application execution time.

The first measure uses two nodes, one master and one worker; for the remaining measures, the number of workers has been increased. Application scalability is good for a small number of CPUs (less than 10), but does not scale so well for greater CPU numbers. However, application behaviour is not greatly affected when using multiple sites. In general, the use of more sites introduces a somewahat larger bigger overhead (around 10 seconds), although it also allows the use of faster CPUs not available in one-site executon. These results show that it is possible to exploit such a Grid environment as a large cluster for executing similar applications, without being limited to the number of CPUs available on a single site.

5 Conclusions

We have described the main components of the resource-management system that we have developed 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 records the job queue submitted by the user and carries out subsequent actions to run the application effectively on the suitable resources. The Resource Searcher has the responsibility of providing groups of machines for any MPI job, taking the application requirements into account. Finally, the Application Launcher is the module that, in the final stage, is responsible for ensuring reliable application execution and co-allocation on the selected resources.

The job-management service provides a reliable on-line and batch MPICH-G2 submission to a Grid. It uses agents to take control of remote machines, allowing the implementation of backfilling scheduling policies for sequential jobs, while all the MPICH-G2 application subjobs are waiting for the proper co-allocation of resources. The Application Launcher guarantees execution without blocking machines, and takes the appropriate decisions in order to guarantee resubmission of failed parallel jobs (due to crashes or failures with the network connection, resource manager or remote resources) and exactly-once execution.

We have tested and evaluated our system, measuring the overhead introduced by the CrossBroker when submitting MPICH-G2 jobs. This overhead is introduced in the case of batch submission and is less than a minute - a short duration for the kind of applications that the batch submission is targeted at,- which usually take much longer to execute. We have also tested system utility by the execution of a master-worker application. This application does not make a heavy use of communications, showing similar scalability both in Grid and in one-site-only execution. Many embarrassingly parallel applications should behave in a similar way, and therefore are also suitable for this environment.

References

1. EU-CrossGrid: http://www.eu-crossgrid.org (2004)
2. Schopt, J.M.: Ten Actions When Grid Scheduling. In: Grid Resource Management - State of the Art and Future Trends. Kluwer Academic Publishers (2003)
3. Karonis, N.T., et alters: Mpich-g2: A grid-enabled implementation of the message passing interface. J. Parallel Distrib. Comput. **63**(5) (2003) 551–563
4. Bucur, A., Epema, D.: The performance of processor co-allocation in multicluster. In: 11th Int. Symp on High Perf. Distr. Comp. (2002)
5. Wang, L., et alters: Resource co-allocation for parallel tasks in computational grids. In: Int. Workshop on Challenges of Large Apps. in Dist. Env. (2003)
6. Czajkowski, K., Foster, I., Kesselman, C.: Resource co-allocation in computational grids. In: Proceedings of the HPDC-8. (1999) 219–228
7. Lindner, P., et alters: Gcm: a grid configuration manager for heterogeneous grid enviromnents. Int. J. Grid and Utility Computing **1**(1) (2005) 4–12
8. Gabriel, E., et alters: Distributed computing in a heterogenous computing environment. In: EuroPVMMPI'98. (1998)
9. Heymann, E., et alters: Managing mpi applications in grid environments. In: European Across Grids Conference. (2004) 42–50
10. Pazini, F.: Jdl attibutes. Technical report, European Datagrid Project (2001)
11. Raman, R., et alters: Matchmaking: Distributed resource management for high throughput computing. In: HPDC-7, Chicago, IL (1998)
12. Thain, D., et alters: Condor and the grid. In: Grid Computing: Making the Global Infrastructure a Reality. John Wiley & Sons Inc. (2003)
13. Gutiérrez, A., et alters: Parallelization of a neural net training program in a grid environment. In: PDP 2004. (2004) 258–265
14. Cameron, D., et alters: Replica management services in the european datagrid project. In: UK e-Science All Hands Conference. (2004)