The EU-CrossGrid Approach for Grid Application Scheduling*

Elisa Heymann[1], Alvaro Fernández[2], Miquel A. Senar[1], 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. This paper presents the approach being followed to implement scheduling components that are integrated as part of the EU CrossGrid project. The purpose of these components is to provide a basis for supporting the efficient execution of distributed interactive applications on Grid environments. When a user submits a job, the scheduling services search for the most suitable resources to run the application and take subsequent steps to ensure a reliable launching of the application. All these actions are carried out according to user-defined preferences.

Introduction

The Grid is an abstraction of distributed heterogeneous systems that has gained much attention in recent years, as shown by current initiatives such as the GriPhyn, iVDT, GrADS, EU-DataGrid, GridLab or EU-CrossGrid projects. The middleware infrastructure provided by these projects will greatly simplify the process of application deployment on computational grids.

In particular, the main objective of the EU CrossGrid project is to take advantage of a collection of machines distributed across Europe, and that constitute a Grid, by making it available to execute user applications. The applications that will use this Grid are mainly parallel and interactive. Although the project will initially benefit from EU Datagrid middleware [1], specific components are currently under development in order to target the concrete requirements of CrossGrid applications. Specifically, new resource-management services will be needed to control the execution of user applications written in MPI in an automatic, reliable and efficient way [2].

* 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.

Within this framework, this paper describes our resource-management approach designed for the CrossGrid project. The paper is organized as follows: Section 2 describes related work, Section 3 describes the overall architecture of our resource management services, Section 4 describes the two of the main services and Section 5 contains the final conclusions to this work.

2. Related Work

There are a number of ongoing research efforts in Grid computing that are considering task allocation problems. Nimrod-G [3] is a tool for the automated modeling and execution of parameter sweep applications (parameter studies) over global computational grids. It provides a simple declarative modeling language for expressing parametric experiments and it includes a resource scheduler that includes policies based on grid economies. Specifically, it requires that each user job have a user-defined deadline and budget constraints for schedule optimizations. With this information, for instance, the broker may try to meet the deadline (completion time) using cheap resources as much as possible. The AppLeS project [4] has been developing schedulers that were generally developed as prototypes to support research into application-level scheduling algorithms for specific applications. However, these schedulers were not general-purpose, to be used for any application. The AppLeS framework guides the implementation of application-specific scheduler logic, which determines and actuates a schedule customized for the individual application and the target compytational Grid environment. Recent efforts in the GrADS project [5][6] try to generalize this approach by decoupling the scheduler core that carries out the search procedure from the application-specific and platform-specific components. The GrADS approach, however, requires a performance model that is an analytical metric for predicting application execution times on a given set of resources.

Datagrid [1] has developed a whole resource management system that was originally targeted to sequential applications. It is based on the Condor ClassAd [7] library for carrying out resource selection and, in the near future, it plans to support MPI applications running on a single cluster. In the CrossGrid resource management, we have adopted the same formalism used in Datagrid to describe jobs [8]. This formalism is derived from Condor ClassAds and is briefly described in the next section. Additionally, our first prototype has been built on the current Datagrid Resource Broker, in order to provide additional support for parallel applications.

Prophet [9] is a run-time scheduling system that, similarly to the AppLeS approach, exploits application structure and system resource information to promote application performance. It was originally demonstrated in heterogeneous, local-area networks, although its concepts could be adapted to Grid environments. It is a scheduler tight with parallel applications written in the Mentat programming language, a fact that limits its usability.

NetSolve [10] is a client-server system that enables users to solve complex scientific problems remotely. The system allows users to solve particular problems by

accessing remote libraries installed across a network. The NetSolve client library is linked in with the user's application and then the application makes calls to NetSolve for specific service. NetSolve searches for computational resources on a network with the appropriate services installed, and runs them with user supplied input parameters. In contrast to the Crossgrid project, Netsolve does not address the problem of arbitrarily submitting user applications to run on a grid environment. Only preinstalled libraries can be accessed through Netsolve, and there is some limited support for parallel applications. It is based on a set of simple API functions and an on adaptive scheduling mechanism for task-farming applications.

3. General Architecture of CrossGrid Resource Management

This section 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 this on grid resources. When users submit their application, our scheduling services will be responsible for optimizing scheduling and node allocation decisions on a user basis. Specifically, they will carry out three main functions:

1. 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.
2. Perform the necessary steps to guarantee the effective submission of the job onto the selected resources. The application is allowed to run to completion.
3. Monitor job 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*) [8], which has been conveniently extended with additional attributes to reflect the requirements of interactive and parallel applications. The main attributes added are the following:

`JobType` Type of executable (this attribute differentiates between sequential and MPI jobs).
`MinNumCPU` Minimum number of CPUs required by the job (required for MPI applications).
`MaxNumCPU` Maximum number of CPUs required by the job (required for MPI applications).
`Priority` Value in the rank 0...20, describing the priority of the job (lower values imply higher priorities; high-priority interactive jobs should have value 0).

The SA selects candidate resources to run the application by sending the Resource Searcher the *JobAd* containing all the requirements and preferences that the job needs

for its execution. The Resource Searcher returns a list with all the combinations of suitable resources available for running the job.

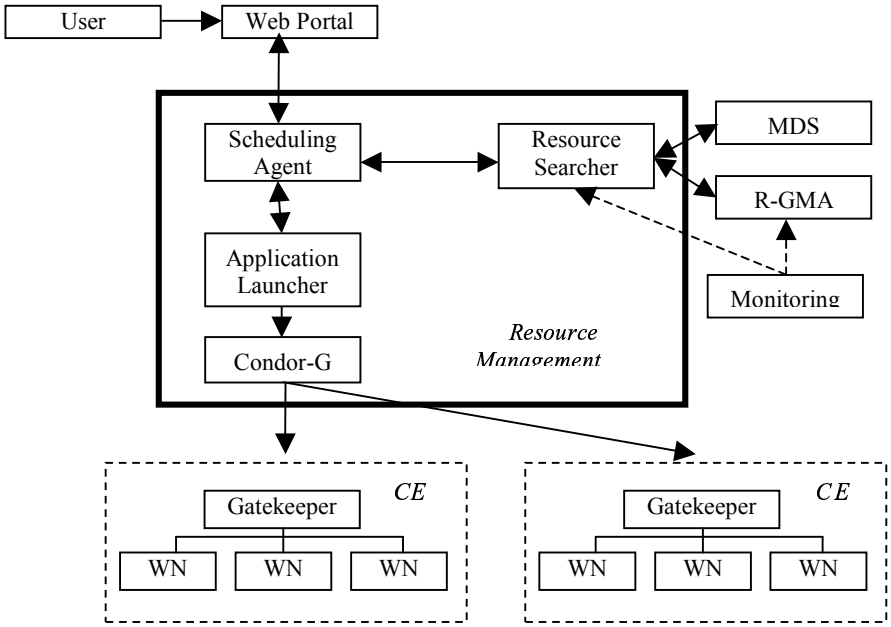

Fig. 1. Resource-Management Architecture.

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 requirements of a particular job 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.

Each 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 [11]. It will also be responsible for managing the available resources in a way that guarantees that high priority jobs (i.e. interactive) are able to run as soon as possible, while lower priority jobs are either temporarily vacated or suspended for a latter resubmission.

Most CrossGrid applications are characterized by interaction with a person in a processing loop. Therefore, an important role to be played by the Scheduling Agent Management is that of taking responsibility for handling job priorities. In our first prototype, priorities will be used to sort the list of pending jobs. In future prototypes, we plan to include both pre-emption mechanisms and temporal-pause mechanisms of low-priority jobs.

The above-mentioned services will be integrated in an overall architecture in which multiple SAs and a single Resource Searcher will exist within a Virtual Organization. SAs (and their Application Launcher and Condor-G counterpart) control user applications on the users' behalf, and there could be one SA-AL-Condor-G set per user. The Resource Searcher informs SAs about the availability of grid resources and only a limited number of these is required. Although our current prototype still exhibits a centralized architecture in which SA and RS are merged in a single entity, by providing a more decentralized set of Scheduling Agents in future prototypes, we hope that EU-CrossGrid resource-management services will avoid both the single point of failure and the performance bottlenecks incurred in centralized solutions, and that they will therefore provide higher reliability.

4. 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 [7].

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 (*suitableCEs*). 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, available physical memory, 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 `MinNumCPU`, as described in the *JobAd*.

In the second step, there are two main structures used for each submitted job:

- *matchedListCEs*: a vector that comprises the groups of CEs that fulfill all the requirements.
- *partiallyMatchedCEs*: a vector of partial groups of aggregated CEs that still do not fulfill all the requirements, but which are suitable candidates if they are aggregated with other suitable CEs.

The selection process goes through the list of single CEs (*suitableCEs*) starting from the beginning. If the evaluated CE fulfils all the requirements, it is included in the *matchedListCEs* (in the first entry, which corresponds to groups with only 1 element), and the process continues with the next entry in *suitableCEs*.

If the evaluated CE does not fulfill the requirements by itself, the matching algorithm goes across the groups of CEs included into *partiallyMatchedCEs* to see whether, by adding this CE to each one of the groups in the vector, a group that fulfils the requirements can be formed. If the new group is found, it is inserted in the corresponding entry of *matchedListCEs* according to the number of elements in the group (second entry, for groups of two CEs; third entry, for groups of three CEs; and so on). If the new CE combined with a group included in *partiallyMatchedCE* does not still fulfill the requirements, the CE is added to that particular group and it is

checked with the next group in the *partiallyMatchedCE* list. Figure 2a and 2b show an example of the results obtained by the Resource Searcher for a job that requires 5 CPUs and wishes to use resource with a better performance in terms of the Average Spect Int benchmark (*AverageSI* in JDL).

```
**********      JDL File      **********
Executable      = "my_app";
JobType         = mpi;
MinNumCpu       = 5;
MaxNumCpu       = 5;
Arguments       = "-n";
StdOutput       = "std.out";
StdError        = "std.err";
Requirements    =  other.OpSys=="RH 6.2";
Rank            =  AverageSI
```

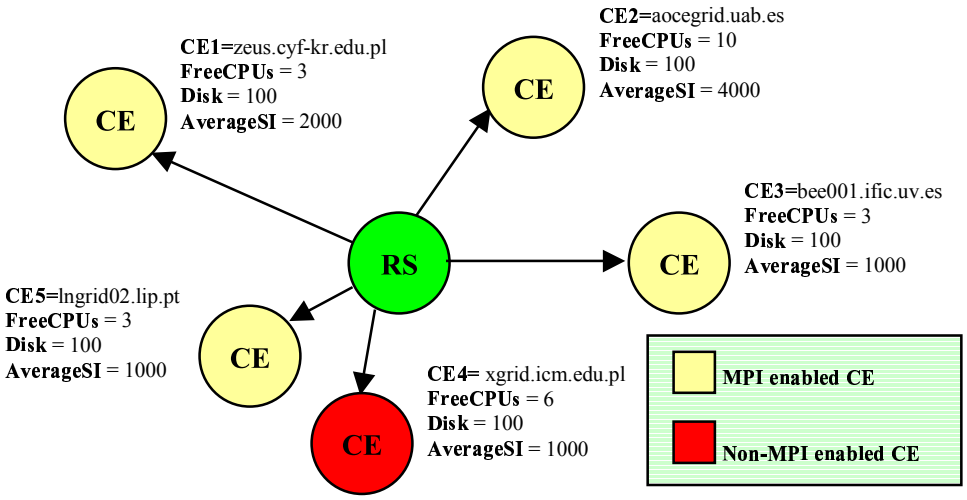

Figure 2a. Example of MPI application submission over CrossGrid sites.

As described above, 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.

In principle, this characteristic should not prevent our Resource Searcher from obtaining good selections. 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 woth 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.

```
[Groups with 1 CEs]
  [Rank=2000]
   aocegrid.uab.es:2119/jobmanager-pbs-workq
   freeCPUs = 10

[Groups with 2 CEs]
  [Rank=1500]
   zeus.cyf-kr.edu.pl:2119/jobmanager-pbs-workq
   freeCPUs = 3
   bee001.ific.uv.es:2119/jobmanager-pbs-workq
   freeCPUs = 3
 [Rank=1000]
    bee001.ific.uv.es:2119/jobmanager-pbs-workq
    freeCPUs = 3
    lngrid02.lip.pt:2129/jobmanager-pbs-workq
    freeCPUs = 3
```

Figure 2.b. Result obtained by the Resource Selector in the submission example of figure 2a.

The Resource Searcher currently supports information collection from the Globus Metacomputing Directory Service (MDS). The MDS is a Grid-information management service that is used to collect and publish system configuration, capability and status information. It is also planned to add support for other resource-information systems, such as R-GMA, from the EU-DataGrid project. R-GMA will be also used to collect information obtained by different monitoring tools currently under development in the CrossGrid project, to track and forecast resource conditions.

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 [11] job management mechanisms. Currently, MPI applications (using the MPICH-G2 implementation, which is a Grid-aware version of MPI) can be started with its own program start-up command (namely, *mpirun*). However, this approach has some limitations with respect to submission reliability and error recovery. The Application Launcher therefore takes advantage of Condor-G, which constitutes a grid-resource manager by exploiting services included in the Globus toolkit, while guaranteeing fault tolerance and exactly-once execution semantics for sequential jobs. The

Application Launcher, based on the Condor-G [6] services for sequential jobs, will provide reliable submission and error-handling mechanisms for MPI applications on the Grid. The Condor-G system is a grid resource manager that provides an API and command line tools that allow the user to perform basic job submission and to obtain access to information about the execution of the job, providing a complete history of their jobs' execution. However, it does not take any scheduling decisions at all, as it requires that actual resources are given directly to it. Condor-G has been designed primarly as a reliable front-end to a computational grid.

It is worth noting that the MPICH-G2 library is currently limited to running on machines that have public IP addresses. This means that the Application Launcher currently under development uses two launching mechanisms: the first mechanism is applied to launch MPICH applications (compiled with the P4 devide) into single clusters in which internal machines have private IPs. The second mechanism is used to launch MPICH applications compiled with the G2 device over multiple nodes that have public IPs. Currently, work is in progress into the Resource Searcher that would allow it to restrict the search between CEs with public or private IPs, according to the kind of MPI application submitted. The jobtype field of the JobAd will be modified accordingly.

Conclusions

We have described the approach that is currently been followed in the EU-CrossGrid in order to provide automatic and reliable support of MPI job management over grid environments. The main components of our job management services are 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. It is based on JobAds as a basic mechanism to represent both jobs and resources, and has been implemented by extending the Condor ClassAd library.

Finally, the Application Launcher is the module that, in the final stage, is responsible for ensuring a reliable starting of the resource application decided by the Scheduling Agent.

Our initial prototype is being built on an EU-DataGrid Resource Broker, which implies that it exhibits a highly centralized architecture. However, work is also in progress to provide an architecture in which multiple SA could connect to a common RS. Additionally, co-operation mechanisms will be also investigated at the level of Resource Searchers, so that queries to one RS could be forwarded to other RSs if insufficient resources were found by the original RS. This research also includes development of mechanisms that prevent potential deadlocks during the actual scheduling and submission of MPI applications over grid resources.

References

1. F. Giacomini, F. Prelz, “Definition of architecture, technical plan and evaluation criteria for scheduling, resource management, security and job description”, DataGrid-01-D1.2-0112-0-
2. F. Giacomini, F. Prelz, M. Sgaravatto, I. Terekhov, G. Garzoglio, and T. Tannenbaum, "Planning on the Grid: A Status Report [DRAFT]", Tech. Rep. PPDG-20, (http://www.ppdg.net), October 2002.
3. Rajkumar Buyya, David Abramson, and Jonathan Giddy, A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker, Future Generation Computer Systems (FGCS) Journal, Elsevier Science, The Netherlands, 2002.
4. H. Casanova, G. Obertelli, F. Berman and R. Wolski, "The AppLeS Parameter Sweep Template: User-level middleware for the Grid. In Proceedings of Supercomputing, November, 2000.
5. Ken Kennedy et alter, “Toward a Framework for Preparing and Executing Adaptive Grid Programs”, Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL, April, 2002.
6. Holly Dail, Henri Casanova, and Fran Berman, “A Decoupled Scheduling Approach for Grid Application Development Environments”, Proceedings of Supercomputing, November, 2002.
7. Rajesh Raman, Miron Livny, and Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", Proc. of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998.
8. 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
9. J. Weissman, "Prophet: Automated scheduling of SPMD programs in workstation networks", Concurrency: Paractice and Experience, 11, 6, 1999.
10. Henri Casanova and Jack Dongarra. "NetSolve: A Network Server for Solving Computational Science Problems. The International Journal of Supercomputer Applications and High Performance Computing", Volume 11, Number 3, pp 212-223, 1997.
11. James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steven Tuecke, "CondorG: A Computation Management Agent for Multi-Institutional Grids", Journal of Cluster Computing, vol. 5, pages 237-246, 2002.
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.