Experiences on Grid Resource Selection Considering Resource Proximity *

Eduardo Huedo[1], Rubén S. Montero[2], and Ignacio M. Llorente[2,1]

[1] Laboratorio de Computación Avanzada, Centro de Astrobiología (CSIC-INTA), 28850 Torrejón de Ardoz, Spain.
[2] Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain.

Abstract. Grids are by nature highly dynamic and heterogeneous environments, and this is specially the case for the performance of the interconnection links between grid resources. Therefore, grid resource selection should take into account the proximity of the computational resources to the needed data in order to reduce the cost of file staging. This fact is specially relevant in the case of adaptive job execution, since job migration requires the transfer of large restart files between the compute hosts. In this paper, we discuss the extension of the Grid*W*ay framework to also consider dynamic resource proximity to select grid resources, and to decide if job migration is feasible and worthwhile. The benefits of the new resource selector will be demonstrated for the adaptive execution of a computational fluid dynamics (CFD) code.

1 Introduction

Grids bring together resources distributed among different administration domains to offer a dramatic increase in the number of available compute and storage resources that can be delivered to applications. The Globus middleware [1] provides the services and libraries needed to enable secure multiple domain operation with different resource management systems and access policies. It supports the submission of applications to remote hosts by providing resource discovery, resource monitoring, resource allocation, and job control services.

However, application execution on grids continues requiring a high level of expertise due to its complex nature. The user is responsible for manually performing all the job submission stages in order to achieve any functionality: resource discovery and selection; and job preparation, submission, monitoring, migration and termination [2]. We have presented in [3] a new Globus experimental framework that allows an easier and more efficient execution of jobs on a dynamic grid environment in a "submit and forget" fashion. Adaptation to changing conditions is achieved by implementing automatic application migration following performance degradation, "better" resource discovery, requirement change, owner decisions or remote resource failure.

* This research was supported by Ministerio de Ciencia y Tecnología (research grant TIC 2003-01321) and Instituto Nacional de Técnica Aeroespacial (INTA).

The most important step in job scheduling is resource selection, which in turn relies completely in the information gathered from the grid. Resource selection usually takes into account the performance offered by the available resources, but it should also consider the proximity between them. The size of the files involved in some application domains, like Particle Physics or Bioinformatics, is very large. Hence the quality of the interconnection between resources, in terms of bandwidth and latency, is a key factor to be considered in resource selection [4]. This fact is specially relevant in the case of adaptive job execution, since job migration requires the transfer of large restart files between the compute hosts. In this case, the quality of the interconnection network has a decisive impact on the overhead induced by job migration.

The architecture of the Grid*W*ay framework and its main functionalities are briefly described in Section 2. In Sections 3 and 4 we discuss the extension of the Grid*W*ay framework to also consider dynamic resource proximity to select grid resources, and to decide if job migration is feasible and worthwhile. The benefits of the new resource selector will be demonstrated in Section 5 for the adaptive execution of a computational fluid dynamics (CFD) code on a research testbed. Finally, in Sections 6 and 7 we describe related and future work, and give some conclusions about the brokering strategy presented in this research.

2 The Grid*W*ay Framework

Probably, one of the most challenging problems that the grid community has to deal with is the fact that grids present unpredictable changing conditions, namely: high fault rate, and dynamic resource availability, load and cost. Consequently, in order to obtain a reasonable degree of both application performance and fault tolerance, a job must be able to migrate among the grid resources adapting itself according to their dynamic characteristics.

The core of the Grid*W*ay framework is a personal *submission agent* that performs all the steps involved in job submission [2]. Adaptation to changing conditions is achieved by supporting automatic job migration. Once a job is initially allocated, it is dynamically rescheduled when the following events occur:

- A "better" resource is discovered
- The remote host or its network connection fails
- The submitted job is cancelled or suspended
- A performance degradation is detected
- The requirements or preferences of the application changed (self-migration)

The architecture of the *submission agent* is depicted in figure 1. The user interacts with the framework through a *request manager*, which handles client requests and forwards them to the *dispatch manager*. The *dispatch manager* periodically wakes up and tries to submit pending jobs to grid resources, it is also responsible for deciding if the migration of rescheduled jobs is worthwhile or not. Once a job is allocated to a resource, a *submission manager* and a *performance monitor* are started to watch over its correct and efficient execution.

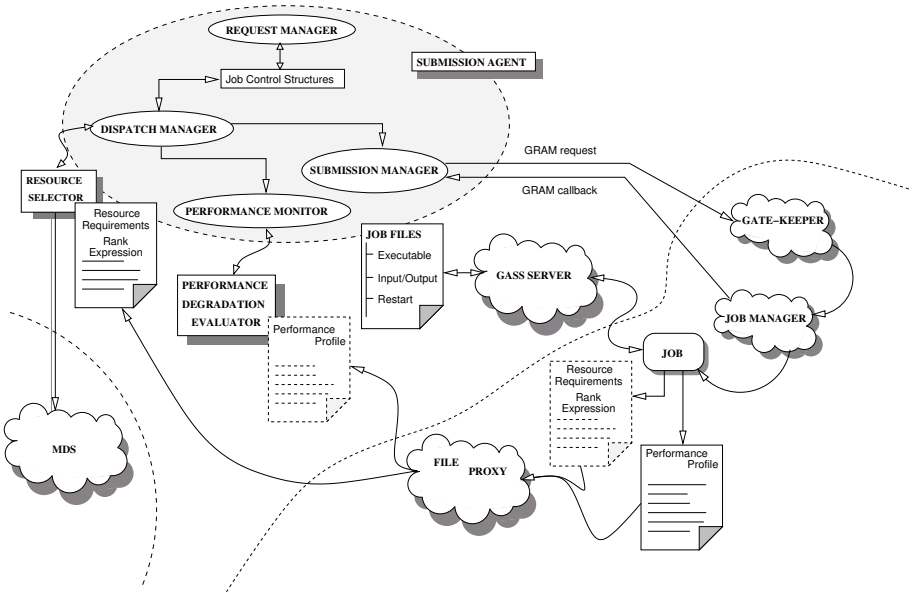

Fig. 1. The architecture of the Grid*W*ay framework.

The flexibility of the framework is guaranteed by a well-defined interface for each *submission agent* component. Moreover, the framework has been designed to be modular, to allow extensibility and improvement of its capabilities. The following modules can be set on a per job basis:

- *resource selector*, which builds a prioritized list of candidate resources
- *performance evaluator*, which is used to evaluate the application performance
- *prolog*, which prepares the remote system and performs input file staging
- *wrapper*, which executes the job and returns its exit code
- *epilog*, which performs output file staging and cleans up the remote system

3 Resource Selection Considering Proximity to Data

Due to the heterogeneous and dynamic nature of the grid, the end-user must establish the requirements that must be met by the target resources (discovery process) and a criteria to rank the matched resources (selection process). The attributes needed for resource discovery and selection must be collected from the information services in the grid testbed, typically Globus MDS. Resource discovery is usually based on static attributes (operating system, architecture, memory size...) taken from MDS GIIS (Grid Information Index Service), while resource selection is based on dynamic attributes (disk space, processor load, free memory...) taken from MDS GRIS (Grid Resource Information Service).

The dynamic network bandwidth and latency between resources will be also considered in the resource brokering scheme. Different strategies to obtain those

network performance attributes can be adopted depending on the services available in the testbed. For example, Globus MDS could be configured to provide such information by accessing the Network Weather Service [5] or by activating the reporting of GridFTP statistics [6]. Alternatively, the end-user could provide its own network probe scripts or static tables.

The brokering process of the Grid*Way* framework is shown in figure 2. Initially, available compute resources are discovered by accessing the MDS GIIS server and, those resources that do not meet the user-provided requirements are filtered out. At this step, an authorization test (via GRAM ping request) is also performed on each discovered host to guarantee user access to the remote resource. Then, the dynamic attributes of each host is gathered from its local MDS GRIS server. This information is used by an user-provided rank expression to assign a rank to each candidate resource. Finally, the resultant prioritized list of candidate resources is used to dispatch the job.

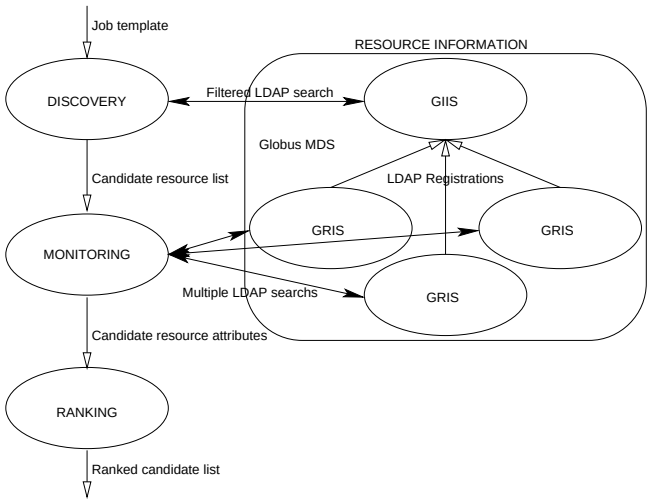

Fig. 2. The brokering process scheme of the Grid*Way* framework.

The new selection process presented in this paper considers both dynamic performance and proximity to data of the computational resources. In particular, the following circumstances will be considered in the resource selection stage:

- The estimated computational time on the candidate host when the job is submitted from the *client* or migrated from other *execution host*.
- The proximity between the candidate host and the *client*, to reduce the cost of job submission, job monitoring and file staging.
- The proximity between the candidate host and a remote *file server*, to reduce the transfer costs of input or output files stored in such server.
- The proximity between the candidate host and the current or last *execution host*, to reduce the migration overhead.

4 Performance Model

In order to reflect all the circumstances described previously, each candidate host (h_n) will be ranked using the *total* submission time (lowest is best) when the job is submitted or migrated to that host at a given time (t_n). In this case, we can assume that the submission time can be split into:

$$T_{sub}(h_n, t_n) = T_{exe}(h_n, t_n) + T_{xfr}(h_n, t_n) \tag{1}$$

where $T_{exe}(h_n, t_n)$ is the estimated computational time and $T_{xfr}(h_n, t_n)$ is the estimated file transfer time.

Let us first consider a single-host execution, the computational time of a CPU-intensive serial application on host h at time t can be estimated by:

$$T_{cpu}(h,t) = \begin{cases} \frac{Op}{FLOPS} & \text{if } CPU(t) \geq 1; \\ \frac{Op}{FLOPS \cdot CPU(t)} & \text{if } CPU(t) < 1 \end{cases} \tag{2}$$

where $FLOPS$ is the peak performance achievable by the host CPU, $CPU(t)$ is the total free CPU at time t, as provided by the MDS default scheme, and Op is the number of floating point operations of the application.

However, the above expression is not accurate when the job has been executing on multiple hosts and then is migrated to a new one. In this situation the amount of computational work that have already been performed must be considered. Let us suppose an application that has been executing on hosts $h_0 \ldots h_{n-1}$ at times $t_0 \ldots t_{n-1}$ and then migrates to host h_n at time t_n, the overall computational time can be estimated by:

$$T_{exe}(h_n, t_n) = \sum_{i=0}^{n-1} t^i_{exe} + \left(1 - \sum_{i=0}^{n-1} \frac{t^i_{exe}}{T_{cpu}(h_i, t_i)}\right) T_{cpu}(h_n, t_n) \tag{3}$$

where $T_{cpu}(h,t)$ is calculated using expression 2, and t^i_{exe} is the time the job has been executing on host h_i, as measured by the framework. Note that, when n is 0, expressions 2 and 3 are equivalent.

Similarly, the following expression estimates the total file transfer time:

$$T_{xfr}(h_n, t_n) = \sum_{i=0}^{n-1} t^i_{xfr} + \sum_j \frac{Data_{h_n,j}}{bw(h_n, j, t_n)} \quad j = client, file\ server, exec\ host \tag{4}$$

where $bw(h_1, h_2, t)$ is the bandwidth between hosts h_1 and h_2 at time t, $Data_{h_1,h_2}$ is the file size to be transferred between them, and t^i_{xfr} is the file transfer time on host h_i, as measured by the framework.

When the job is rescheduled due to a resource discovery timeout, the *dispatch manager* chooses the lowest ranked (lowest estimated submission time) candidate host for a migration only if it has a lower rank than the rank of the current *execution host* when the job was submitted; otherwise the migration is rejected. Therefore, the job will be migrated only if migration is considered to be

profitable. However, when the job is rescheduled due to a remote host failure or an user request, the *dispatch manager* always grants the migration to the lowest ranked host, even if it has a higher rank than the last *execution host*.

5 Experiments

The behavior of the resource selection strategy previously described is demonstrated in the execution of a CFD code in our heterogeneous research testbed, summarized on Table 1. The target application solves the 3D incompressible Navier-Stokes equations using an iterative multigrid method [7].

Table 1. Characteristics of the machines in the research testbed.

Name	Model	OS	Speed	Memory
ursa.dacya.ucm.es	Sun Blade 100	Solaris 8	500MHz	256MB
draco.dacya.ucm.es	Sun Ultra 1	Solaris 8	167MHz	128MB
columba.dacya.ucm.es	Intel Pentium MMX	Linux 2.4	233MHz	160MB
cepheus.dacya.ucm.es	Intel Pentium Pro	Linux 2.4	200MHz	64MB
solea.quim.ucm.es	Sun Enterprise 250	Solaris 8	296MHz	256MB

In the following experiments, the *client* is ursa, which holds an input file with the simulation parameters, and the *file server* is columba, which holds the executable and the computational mesh. The output file with the velocity and pressure fields is transferred back to the *client*, ursa, to perform post-processing. Table 2 shows the available machines in the testbed, their corresponding CPU performance (in MFLOPS), and the maximum bandwidth (in MB/s) between them and the hosts involved in the experiment.

Table 2. Candidate machines in the testbed along with their CPU performance and bandwidth between them and the machines involved in the experiment (*client*=ursa, *file server*=columba and *exec host*=draco).

h	CPU	$bw(h, client)$	$bw(h, file\ server)$	$bw(h, exec\ host)$
draco	175	0.4	0.4	∞
columba	225	0.4	∞	0.4
cepheus	325	0.4	0.4	0.4
solea	350	0.2	0.2	0.2

Table 3. Estimated and measured times for a complete execution without migration.

h_0	$T_{exe}(h_0,t_0)$	$T_{xfr}(h_0,t_0)$	$T_{sub}(h_0,t_0)$	Measured time
draco	171	20	191	200
columba	133	10	143	146
cepheus	92	20	**112**	**120**
solea	**86**	40	126	134

Table 4. Estimated and measured times for a migrated execution from draco (h_0=draco, $T_{exe}(h_0,t_0)$=171, t^0_{exe}=65, t^0_{xfr}=10).

h_1	$T_{exe}(h_1,t_1)$	$T_{xfr}(h_1,t_1)$	$T_{sub}(h_1,t_1)$	Measured time
columba	148	30	178	184
cepheus	122	40	**162**	**170**
solea	**118**	70	188	196

Table 3 shows the estimated submission time (1), computational time (3) and transfer time (4) for a single-host execution on each of the available hosts. Resource selection based only on the estimated computational time would allocate the application to solea (lowest T_{exe}). However, when the resource selection strategy takes into account the file transfer time, the job is allocated to cepheus (lowest T_{sub}). The measured submission time presents a reduction of about 10% compared with the resource selection based only on CPU performance.

The influence of the file transfer size is even more relevant in case of an adaptive execution, as shown in Table 4. In this case a migration from draco to each of the available hosts in the testbed is evaluated. When only the CPU performance is considered in the resource selection process, the job is migrated to solea (lowest T_{exe}). However, if resource proximity is also considered, the application would migrate to cepheus (lowest T_{sub}) that implies a reduction of the submission time of about 14%, compared with the resource selection based only on CPU performance.

6 Related Work

The selection stage of computational resources has been widely studied in the literature. For example, the resource selection service presented in [8] uses an extension of the Condor matchmaking algorithm called set matching to provide selection of resource sets. However, it is focused on distributed HPC applications and it only takes into account the proximity between the resources within a resource set. Also, a framework capable to migrate applications based on load conditions is proposed in [9], although it considers the migration overhead as

a constant. Finally, several mechanisms for storage resource selection [10] and data replication [11] based on proximity have been proposed. We apply some of these ideas to the selection of computational resources, taking into account the possibility of job migration to deal with the dynamic nature of the grid.

7 Conclusions and Future Work

In this work we have analyzed the relevance of resource proximity in the resource selection process, in order to reduce the cost of file staging. In the case of adaptive job execution the quality of the interconnection network has also a decisive impact on the overhead induced by job migration. In this way, considering resource proximity to the needed data is, at least, as important as considering resource performance characteristics. We expect that resource proximity would be even more relevant for greater file sizes and more heterogeneous networks.

We are currently applying the same ideas presented here to develop a *storage resource selector* module. The storage resource selection process is equivalent to the one presented in figure 2, although the discovery process is performed by accessing the Globus Replica Catalog. The resource selection is based on the bandwidth between the selected compute resource and the candidate storage resources, along with the values gathered from the MDS GRIS.

References

1. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. J. of Supercomputer Applications **11** (1997) 115–128
2. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, The Global Grid Forum: Scheduling Working Group (2001)
3. Huedo, E., Montero, R.S., Llorente, I.M.: A Framework for Adaptive Execution on Grids. Intl. J. of Software – Practice and Experience (2004) (in press).
4. Allcock, W., et al.: Globus Toolkit Support for Distributed Data-Intensive Science. In: Proc. of Computing in High Energy Physics. (2001)
5. Wolski, R., Spring, N., Hayes, J.: The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. J. of Future Generation Computing Systems **15** (1999) 757–768
6. Vazhkudai, S., Schopf, J., Foster, I.: Predicting the Performance of Wide-Area Data Transfers. In: Proc. of Intl. Parallel and Distributed Processing Symp. (2002)
7. Montero, R.S., Llorente, I.M., Salas, M.D.: Robust Multigrid Algorithms for the Navier-Stokes Equations. Journal of Computational Physics **173** (2001) 412–432
8. Liu, C., Yang, L., Foster, I., Angulo, D.: Design and Evaluation of a Resource Selection Framework for Grid Applications. In: Proc. of the Symp. on High-Performance Distributed Computing. (2002)
9. Vadhiyar, S., Dongarra, J.: A Performance Oriented Migration Framework for the Grid. In: Proc. of the Intl. Symp. on Cluster Computing and the Grid. (2003)
10. Vazhkudai, S., Tuecke, S., Foster, I.: Replica Selection in the Globus Data Grid. In: Intl. Workshop on Data Models and Databases on Clusters and the Grid. (2001)
11. Lamehamedi, H., Szymanski, B.K., Deelman, E.: Data Replication Strategies in Grid Environments. In: Proc. of 5th Intl. Conf. on Algorithms and Architectures for Parallel Processing. (2002)