Resource Performance Management on Computational Grids. *

O. San José[†] L. M. Suárez[†] E. Huedo[‡] R. S. Montero[†] I. M. Llorente[†,‡]

[†]Dpto. Arquitectura de Computadores y Automática
Facultad de Informática, Universidad Complutense
28040 Madrid (Spain)

[‡]Lab. Computación Avanzada, Simulación y Aplicaciones Telemáticas
Centro de Astrobiología (CSIC-INTA)
28850 Torrejón de Ardoz (Spain)

Abstract

The ability to have applications draw computing power from a global resource pool to achieve high performance has become a new challenge for distributed computing and Internet technologies. This challenge not only involves solving technical difficulties in the construction of Grid environments, it also involves resource sharing and performance concerns. This paper presents a resource performance manager that fits in the current Globus implementation of computational Grids, and shows how it can aid in the Grid's expansion, illustrated by some experiments carried out on the UCM-CAB testbed.

1. Introduction

For certain application domains the traditional concept of computing based on a homogeneous, and centrally managed environment is being displaced by a new model based on the exchange of information and the sharing of distributed resources by applications [7]. However, such applications often involve large amounts of data and/or computing elements and are not easily handled by today's Internet and web infrastructures. Grid technologies attempt to provide the support needed for such an infrastructure, enabling applications to use remote resources managed by widespread "virtual organizations". The Globus project has constructed an open-source toolkit [6] to build computational Grids, implementing a set of non-proprietary protocols for securely identifying, allocating and releasing resources from the Grid.

* This research was supported by Ministerio de Ciencia y Tecnología, through the research grant TIC 2002-00334, and Instituto Nacional de Técnica Aeroespacial (INTA).

Due to its open-source nature and its increasing popularity, the Globus toolkit has become a *de facto* standard in Grid computing. Globus is a core Grid middleware that provides the following components, which can be used separately or altogether, to support Grid applications: GRAM (Globus Resource Allocation Manager), GASS (Global Access to Secondary Storage), GSI (Grid Security Infrastructure), MDS (Monitoring and Discovery Service), and GridFTP. These services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [12].

The Grid not only involves the technical challenge of constructing and deploying this vast infrastructure, it also brings up other issues related to resource-sharing policies. Undoubtedly, a tool that allows administrators to have full control of their resources should help solve these socio-political difficulties [13]. This should not be regarded as a cutback to the Grid's expansion. On the contrary, the more confident resource owners are, the more nodes they will add to the Grid, overcoming the typical scenario where administrators will share only a small fraction of their hosts due to their mistrust on the Grid.

This work focuses on an addition to the resource management pillar, providing a tool for system administrators to determine the amount of resources they are willing to devote to the Grid, avoiding their saturation by Grid jobs.

The next section outlines the steps involved in the submission of a Grid job and the way the Grid Resource Performance Manager (GRPM) fits in this architecture to filter job requests transparently. Following this, section 3 describes how this model adjusts in a non-intrusive way with general Grid schedulers. A detailed view of the tool's architecture is given in section 4, describing the two components that comprise the performance manager, namely the wrapper and the system monitor. In section 5 we show the tool at work on the UCM-CAB testbed. The paper ends in section 6 with some conclusions.

2. Globus Resource Allocation Manager

GRAM (see figure 1) is the core of the Resource Management pillar of the Globus Toolkit. When a job is submitted, the request is sent to the *gatekeeper* of the remote computer. The *gatekeeper* is a service running on every node of a Globus Grid. The *gatekeeper* handles each request and creates a *job manager* for each job, mutually authenticating with the client, and mapping the request to a local user. The *job manager* starts and monitors the job according to its RSL specification [2], communicating state changes back to the GRAM client via callbacks. When the job terminates, either normally or by failing, the *job manager* terminates as well, ending the life cycle of the Grid job.

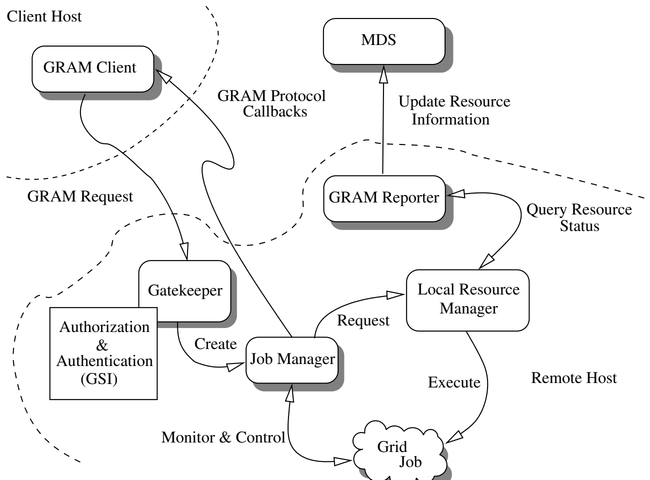

Figure 1. Architecture of the Globus Resource Allocation Manager (GRAM).

GRAM operates in conjunction with a number of schedulers including Condor, PBS and a simple ”fork” scheduler. The *job manager* provides a plugin architecture for extensibility. When the *job manager* is created by the *gatekeeper* to process the job request, it maps the request to a local scheduler. These plugins provide a set of programs and scripts that map job manager requests to scheduler commands such as queue, submit, poll and remove.

GRPM, like GRAM, relies heavily on the interaction with these job schedulers through scripts. A further discussion of these scripts is given once the GRPM tool is introduced. Support for a new scheduler by GRPM requires only the implementation of the relevant scripts, many of which can be taken directly from Globus.

The main drawback of this client/server model, which GRPM tackles, is that Globus doesn't provide any mechanism to deny job requests under stress conditions. Once the *job manager* spawns the Grid job, the job behaves like any other process, competing for the host's resources such as CPU time, physical or swap memory, or I/O devices, inevitably degrading the overall throughput. Many system administrators would like to regard remote Grid jobs as being guest applications that are welcome to use local resources as long as the overall performance remains above certain configurable thresholds.

This architecture makes it adequate to provide a wrapper for the *gatekeeper* service, rejecting or allowing job requests according to the current work load. In Grid environments, resources are constantly being added, removed or temporarily declared out of service. It is therefore evident that Grid based frameworks must tolerate and respond adequately to these inconveniences, including those caused by the denial of a job request by the GRPM tool (see discussion in the next section).

3. General Architecture of Grid Schedulers

Grid scheduling or superscheduling [7], has been defined in the literature as the process of scheduling resources over multiple administrative domains. In general, this process includes the following phases: system selection and preparation; and job submission, monitoring, migration and termination [12]. Therefore, to coexist with any Grid scheduling system the design of a performance management architecture must consider the phases of the scheduling process.

In the following list we analyze the Grid scheduling steps that could be affected by the performance management architecture proposed in this paper:

- *Resource Discovery and Selection:* This step involves the selection of a resource (or resource set) from a list of candidate resources, all of which meet the application specific requirements. An authorization filter is also performed in this phase to guarantee user access to each candidate resource. GRPM will deny connections to the GRAM *gatekeeper* when a system-defined threshold has been exceeded, failing any authentication/authorization attempt. Therefore any overloaded resource will be transparently excluded from the candidate resource list. Moreover, most of the Grid scheduling systems access a Grid information service (MDS, Network Weather Service [15]...) to prevent scheduling jobs on overloaded resources [11, 14, 8, 3, 4, 5, 10]. However the information published by these services may be obsolete. In this way, GRPM protects resource performance from both degradation or absence of Grid information, and *selfish* scheduling systems.

- *Job Monitoring*: As explained in the previous section, job state transitions are notified to the client by the *job manager*. GRPM can suspend Grid jobs to preserve resource performance and resume them when the resource becomes idle. The *job manager* will notify these changes to the Grid scheduling system, ensuring that the appropriate scheduling decision can always be taken, for example re-scheduling the job if it has been suspended by GRPM for a long time.
- *Job Termination*: In order to provide detailed error information (job exit code, system call or Globus error codes...) most of the Grid scheduling frameworks [4, 1, 3, 9] use a *job wrapper* to execute the actual job. GRPM may cancel Grid jobs when a system-defined threshold has been exceed to prevent resource performance degradation. This will lead to an undefined job exit code, that could be used by the Grid scheduler [9] to detect the job cancellation.

4. Grid Resource Performance Manager

The architecture of the Grid Resource Performance Manager (GRPM), depicted in figure 2, is divided into two modules: GRPM wrapper (GRPMW) and GRPM system monitoring and administration daemon (GRPMD). GRPMW filters out Grid requests attending to the current state of the system. The other module, GRPMD, continously monitors the system, proceeding to cancel, suspend or resume Grid jobs if system resources satisfy certain conditions. In this way, GRPM can assure certain minimum computational resources devoted exclusively to local users, even in the worst stress conditions.

The philosophy used in the implementation of GRPM is the one followed by the Globus project. The main feature of this architecture is the use of system-dependant plugins to accomplish the tasks that require an interaction with the system, or the local *job manager*. This modular architecture enables an easy deployment of GRPM in an heterogeneous Grid, and an easy adaptation to a system using either GRPM-, Globus-supplied or custom scripts.

4.1. GRPM Wrapper

The GRPM wrapper is placed between the Globus *gatekeeper* and Grid clients. It replaces Globus *gatekeeper* service at the Internet daemon (`inetd`) level, so Grid connections will be handled by it before reaching the Globus *gatekeeper*, rejecting these connections when the system is overloaded.

Since different administrators are willing to devote different amounts of resources to Grid, and every system is different in performance and availability, GRPMW allows administrators to configure the desired behavior of the tool.

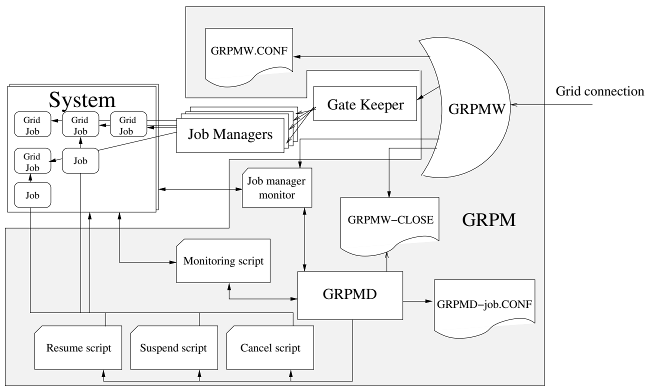

Figure 2. Architecture of the Grid Resource Performance Manager (GRPM).

This is accomplished by supplying an interface to the local *job manager*, and by configuring the threshold that will filter out incoming connections.

The former consists in writing (or choosing) a script (`JOB_MANAGER_MONITOR`) that will provide GRPMW with access to the lists of processes/jobs and users currently on the system. The way of obtaining the current list of users and jobs may vary depending on the system and local job manager. The latter requires administrators to determine the value of the threshold variables, namely:

- `MAXUSERS`: Maximum number of users log on to the system.
- `MAXGRIDUSERS`: Maximum number of Grid users, those registered in the `gridmap-file`.
- `MAXJOBS`: Maximum number of running jobs belonging to local users.
- `MAXGRIDJOBS`: Maximum number of running jobs belonging to Grid users.

These variables will hold the maximum values up to which GRPMW will continue accepting Grid connections (thereby new jobs). All that is required is to assign them an integer value in the configuration file (`grpmw.conf`), but in order to optimize local/Grid ratio of resource performance administrator must choose them carefully.

Note that this architecture enables assigning different policies to the job managers of a given resource. Typically, clusters use fork at the cluster front-end for file transfer operations, while job execution is handled using the batch system job manager [9] (PBS, Condor, SGE...). Therefore, access to the front-end (`jobmanager-fork`) could be more restrictive than the access to the batch system (`jobmanager-pbs`, for example).

Every time a GRAM request is received, the wrapper proceeds to gather some information on the system through the `JOB_MANAGER_MONITOR`, and evaluates the connection according to the following circumstances:

1. The threshold has been exceeded (this means the actual value of any of the threshold variables described above exceeds the maximum allowed).
2. The connection has been closed by an external source, such as GRPMD or the system administrator. This is done through the creation of a 'lock file' (`grpmw-close`).

In either of these two cases, the connection will be rejected, and the client will receive a GRAM related error. Otherwise, the connection will be forwarded to the Globus *gatekeeper*, which will begin its normal operation. A summary of GRPMW's operation flow is shown in figure 3.

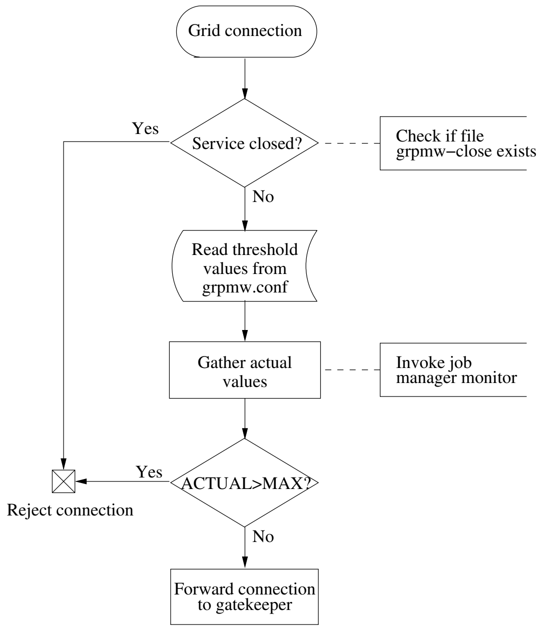
Grid connection
Yes
Service closed?
Check if file grpmw−close exists
No
Read threshold values from grpmw.conf
Gather actual values
Invoke job manager monitor
Yes
ACTUAL>MAX?
Reject connection
No
Forward connection to gatekeeper

Figure 3. Grid Resource Performance Manager Wrapper operation flow.

4.2. GRPM System Monitoring and Administration Daemon

GRPMW alone is not enough to handle all the situations that may lead to a lack of resources due to excessive Grid computing demands or growing requirements of local users. Local users can be given priority over Grid users in a variety of ways, usually platform and *job manager* dependant, such as user or group policies. However this is non-trivial and not easily portable, which can be a problem, especially in a large-sized domain. GRPMD offers a homogeneous, easily-configurable, deployable and decentralized-managed way of administrating these priorities.

The system monitoring and administration daemon (GRPMD) is periodically monitoring the system and, when some system-level defined conditions are satisfied, it proceeds to suspend, cancel or resume some Grid jobs, to allocate resources to real-time needs of local users and system capabilities. Administrators must configure the tool through a set of variables. These variables are divided into two groups: script-location variables, and configuration variables.

The former are paths to the *job manager* plugins that will provide GRPMD with access to the following:

1. Basic operations of local *job manager* (such as suspend, cancel or resume jobs). Since this is also the way Globus middleware issues commands to a given system, administrators can use the scripts shipped with Globus, custom scripts or the ones provided with GRPM.
2. An interface for retrieving information from the system (`SYSTEM_MONITOR`). This is a basic part of the daemon, since it will give GRPMD a snapshot of the state of the system at a given time. Since Globus uses GRAM reporter scripts to gather information associated with each job manager, administrators can use them to gather information from the system. All this information exchange is done in text mode using the following format:

$$\begin{array}{l} BEGIN \\ variable_1 = value_1 \\ \cdots \\ variable_n = value_n \\ END \end{array}$$

   where $variable_i$ could be, for example, the number of users and processes currently on the system, the amount of free memory and swap space, or the CPU usage. Every administrator can use the variables considered appropriate, according to system specific metrics. The only requisite is that a particular variable must appear and be given a value in the monitoring script.

Triggering expressions are the way administrators will tell the daemon when they want it to intervene. They are expressions involving comparisons between the monitor variables (provided by the `SYSTEM_MONITOR`) and values, and boolean operators (with a C-like syntax). On each monitoring interval, GRPMD will read the monitor variables, and will use their values to evaluate the triggering expressions.

If a particular expression evaluates to true, GRPMD will use the associated script to manipulate a predefined number of Grid jobs, trying to suspend, cancel or resume them. In particular, these expressions are:

1. `CANCEL`: if the cancel expression is satisfied, GRPMD starts cancelling Grid jobs at each monitoring interval until the cancel condition evaluates to false. Also, GRPMD creates the `grpm-close` file to reject any new GRAM request.
2. `SUSPEND`: similarly when the suspend triggering expression is satisfied Grid jobs are suspended at each monitoring interval. In this case GRPMD also creates the `grpm-close` file.
3. `RESUME`: if none of the above conditions are satisfied any suspended Grid job is resumed when the resume triggering expression evaluates to true. In this situation the `grpm-close` file is removed.

This mechanism allows every administrator to focus daemon's behavior on resources considered critical in the system's performance. This is done by writing an expression based on variables that represents the state of that resource, and making a monitoring script capable of assigning them a value based on the actual state of that resource.

Resource	Model	Speed (MHz)	Mem. (MB)	Nodes
ursa	Sun Blade 100	500	256	1
draco	Sun Ultra 1	167	128	1
pegasus	Intel Pentium 4	2400	1024	1
solea	Sun Enterp. 250	296	256	2
babieca	Alpha DS10	466	1024	4

Table 1. Summary of the UCM-CAB grid resource hardware characteristics.

Resource	OS	GRAM	VO
ursa	Solaris 8	fork	UCM
draco	Solaris 8	fork	UCM
pegasus	Linux 2.4	fork	UCM
solea	Solaris 8	fork	UCM
babieca	Linux 2.4	PBS	CAB

Table 2. Summary of the UCM-CAB grid resource software characteristics.

5. Experiences

We next demonstrate the capabilities of GRPM when managing the performance of a Grid resource. In particular, we are interested in preserving the CPU performance for the owner of a workstation (pegasus) in the testbed. The configuration files for the GRPM wrapper and system monitor are as follows:

1. The GRPM wrapper only allows one simultaneous Grid user to use the system (`MAX_GRID_USERS = 1`).
2. The GRPM system monitor guarantees that the CPU usage is devoted to pegasus local users when they require it, thus it cancels Grid jobs when the local CPU load increases (`CANCEL = local_cpu_load > 25`). Once the system is no longer used by local users, GRPM accepts Grid requests again (`RESUME = local_cpu_load < 15`).

A parameter sweep application consisting of 50 independent tasks is executed in the testbed. Each task calculates the flow over a flat plate for a different Reynolds number, ranging from 10^2 to 10^4. The experiment files consists of the executable (2MB), and the computational mesh (0.5MB), provided for all the resource architectures in the testbed. Once the job finishes, the standard output (8KB) and the velocity profile (5KB) at the middle of the flat plate are transferred back to the client. The experiment was conducted on the UCM-CAB testbed (see tables 1 and 2), using the Grid*W*ay [9] framework to schedule the 50 jobs.

Let us first consider the performance obtained when the testbed was fully available. In this case, the overall execution time for parameter sweep application was 2376 seconds, with an average job turnaround time of 47.5 seconds (see figure 4, upper chart).

This experiment was repeated introducing an artificial workload in pegasus during 815 seconds at the middle of the execution. As can be expected, while GRPM removed pegasus from the testbed a degradation in the job turnaround was observed (58.4 seconds) and the total execution time increased to 2662 seconds. Once pegasus becomes available again the average job turnaround time decreases to 52.4 seconds.

Figure 4, lower chart, shows the CPU load registered in pegasus during the experiment. From the beginning, to time step 1100, pegasus executed seven grid tasks, clearly shown by the file transfer and job execution times. At time step 1100 the artificial workload decreased the performance of both the grid task and local process. The GRPM monitors detected this situation by firing the cancel triggering expression, aborting the grid job that just entered the system and rejecting further connections. Pegasus executed the

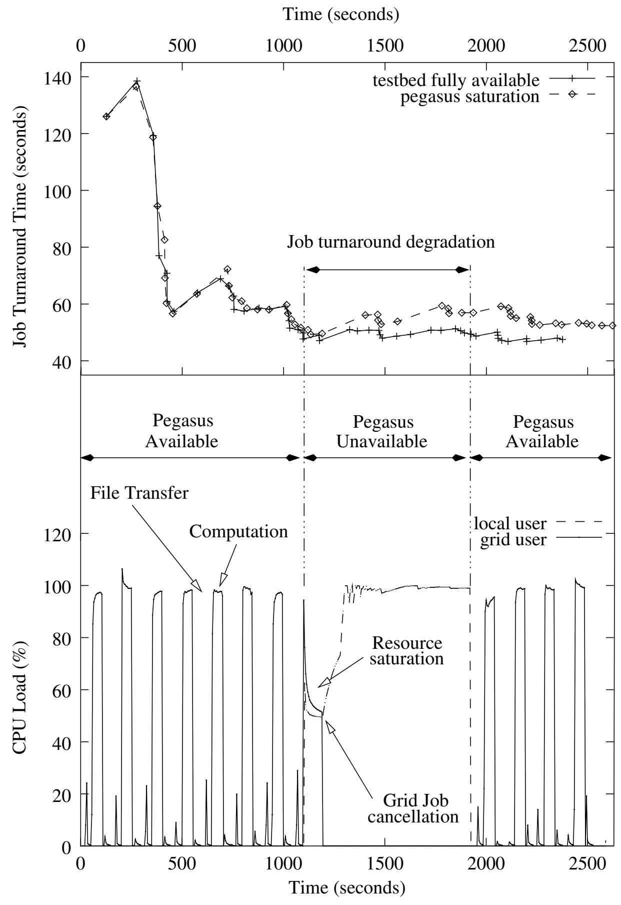

Figure 4. Dynamic job turnaround time in the execution of the parameter sweep application when the UCM-CAB testbed is fully available and when pegasus is saturated (upper chart). CPU load on pegasus induced by local and grid processes (lower chart).

local job until its termination on time step 1920. At that point the CPU became idle so Grid connections were accepted again, and four more jobs entered the system until the moment the experiment ended.

During the time the GRPM wrapper rejected connections, the Grid*W*ay framework adaptively scheduled the pending jobs to other Grid resources. In particular, the canceled job was dynamically rescheduled by the Grid*W*ay framework, and submitted to babieca (see figure 5).

6. Conclusions

The tests carried out demonstrate that the performance manager behaves as expected and doesn't interfere with Grid schedulers in any of the possible situations that can occur: connection forwarding or denial, and job suspension or

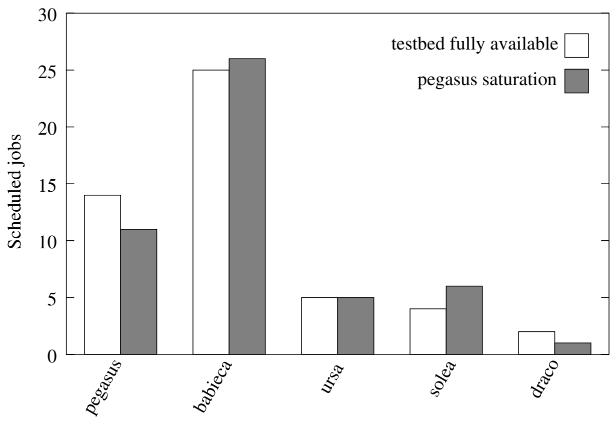

Figure 5. Number of jobs scheduled on each resource when the testbed is fully available, and when pegasus is saturated.

cancellation. As seen, the tool is easily configurable and deployable due to the scripting interface it provides.

These reasons, together with the rest of its features make GRPM a good choice for administrators to configure their Grid infrastructure. We hope GRPM will aid in the expansion of the Grid, leading users to embrace Grid technologies and share their resources with more confidence, because their performance will always be assured.

References

[1] Comparison of HTB and Nimrod/G: Job Dispatch. Available at http://www-unix.mcs.anl.gov/∼giddy/comp.html.

[2] Globus Resource Allocation Manager (gram 1.6) Documentation. Available at http://www-unix.globus.org/api/c-globus-2.2/globus_gram_documentation/html/, 2002.

[3] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid. In *Proceedings of the 4th IEEE International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia)*, 2000.

[4] S. Cavalieri and S. Monforte. Resource Broker Architecture and APIs. Available at http://server11.infn.it/workload-grid/documents.html, 2001.

[5] H. Dail, H. Casanova, and F. Berman. A Decoupled Scheduling Approach for the GrADS Program Development Enviroment. In *Proc. of the SuperComputing (SC02)*, 2002.

[6] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. Supercomputer Applications*, 11(2):115–128, 1997.

[7] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, 1999.

[8] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor/G: A Computation Management Agent for Multi-Institutional Grids. In *Proc. of the 10th Symp. on High Performance Distributed Computing (HPDC10)*, August 2001.

[9] E. Huedo, R. S. Montero, and I. M. Llorente. A Framework for Adaptive Execution in Grids. *Intl. J. of Software – Practice & Experience*, 2003. To appear.

[10] C. Liu, L. Yang, I. Foster, and D. Angulo. Design and Evaluation of a Resource Selection Framework for Grid Applications. In *Proc. of the 11th IEEE Symposium on High-Performance Distributed Computing*, 2002.

[11] R. S. Montero, E. Huedo, and I. M. Llorente. Grid Resource Selection for Opportunistic Job Migration. In *Intl. Conf. on Parallel and Distributed Computing (Euro-Par)*, volume 2790 of *Lecture Notes on Computer Science*, pages 366–373. Springer-Verlag, August 2003.

[12] J. M. Schopf. Ten Actions when Superscheduling. Technical Report WD8.5, The Global Grid Forum, 2001. Scheduling Working Group.

[13] J. M. Schopf and B. Nitzberg. Grids: The Top Ten Questions. Available at http://www-unix.mcs.anl.gov/∼schopf, 2002. To appear in Scientific Programming, special issue on Grid Computing.

[14] S. Vadhiyar and J. Dongarra. A Performance Oriented Migration Framework for the Grid. In *Proc. of the 3rd Intl. Symp. on Cluster Computing and the Grid (CCGrid)*, 2003.

[15] R. Wolski, N. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, 15(5-6):757–768, 1999.