Available online at www.sciencedirect.com
ScienceDirect
ELSEVIER
Future Generation Computer Systems 23 (2007) 252–261
FGCS
FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

A modular meta-scheduling architecture for interfacing with pre-WS and WS Grid resource management services ☆

Eduardo Huedo*, Rubén S. Montero, Ignacio M. Llorente

Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense, 28040 Madrid, Spain

Received 20 October 2005; received in revised form 17 July 2006; accepted 20 July 2006
Available online 1 September 2006

Abstract

The last version of the Globus Toolkit includes both pre-WS and WS GRAM services to submit, monitor, and control jobs on remote Grid resources. In the medium term and until a full transition is accomplished, both pre-WS and WS GRAM services will coexist in Grid infrastructures. In this paper, we describe the modular architecture of the Grid*W*ay meta-scheduler, which allows the simultaneous and coordinated use of pre-WS and WS GRAM services and, therefore, makes easy the transition to a Web Service implementation of the Globus components. Such functionality is demonstrated on a infrastructure that comprises resources from a research testbed, based on the Globus Toolkit 4.0, and the EGEE production infrastructure, based on the LCG middleware. The Web Service implementation of Globus components has been optimized for flexibility, stability and scalability. However, part of the Grid community is still reluctant to transition to the Web Service model due mainly to its supposed lower performance. We demonstrate that WS GRAM achieves a performance comparable to that of pre-WS GRAM.

Keywords: Grid computing; Grid*W*ay Meta-scheduler; Globus toolkit; Grid resource management; WSRF

1. Introduction

The main driving force behind moving from pre-WS to WS (Web Services) Grid services is that, according to the Grid's second requirement proposed by Foster [1], a Grid must be built using standard, open, general-purpose protocols and interfaces. However, a large part of the Grid community is still reluctant to make this transition because of the lower efficiency associated with Web Services. In fact, the Grid's third requirement is that a Grid must deliver nontrivial qualities of service, in terms of response time, throughput, security, reliability or the coordinated use of multiple resource types.

On the one hand, pre-WS Grid services are based on proprietary interfaces (although usually implemented over standard protocols like HTTP, LDAP or FTP). On the other hand, WS Grid services are based on the *WS-Resource Framework* (WSRF) [2], a standard specification fully compatible with other Web Service specifications. In fact, WSRF can be viewed as a set of conventions and usage patterns within the context of established Web Service standards, like WS-Addressing. WSRF defines the WS-Resource construct as a composition of a Web service and a stateful resource [3].

The *Open Grid Services Infrastructure* (OGSI) [4] was previously conceived as an extension of Web Services to have stateful WS-Resources. However, the implementation of OGSI resulted in non-standard, complex and heavy-weight Grid services. Moreover, it jeopardized the convergence of Grid and Web Services. Grid services implemented as Web Services are easier to specify and, therefore, to standardize. Thus, WS Grid services provide a way to construct an *Open Grid Services Architecture* (OGSA) [5] where tools from multiple vendors interoperate through the same set of protocols and interfaces, implemented in different manners and over different tools.

As the introduction of OGSA and the release of the *Globus Toolkit* 4 (GT4) have made imminent the generalized use of WS Grid services; it is highly interesting to evaluate

☆ This research was supported by Consejería de Educación de la Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE), through BIOGRIDNET Research Program S-0505/TIC/000101, and by Ministerio de Educación y Ciencia, through the research grant TIC2003-01321. The authors participate in the EGEE project, funded by the European Union.

* Corresponding author. Tel.: +34 91 394 75 38; fax: +34 91 394 75 27.
E-mail addresses: ehuedo@fdi.ucm.es (E. Huedo), rubensm@dacya.ucm.es (R.S. Montero), llorente@dacya.ucm.es (I.M. Llorente).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.07.013

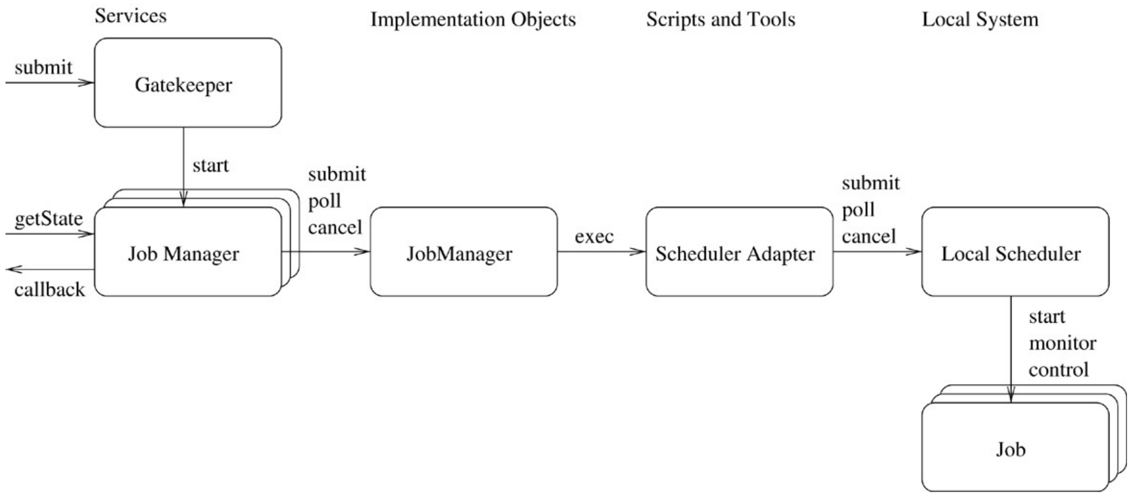


Fig. 1. Architecture of the pre-WS *Grid Resource Allocation and Management* (GRAM) service.

the performance penalty for achieving the Grid's second requirement. In particular, our interest is focused on the main component of Globus-based computational Grids, the *Grid Resource Allocation and Management* (GRAM) service.

In this work we evaluate the GT4 GRAM services from the user's perspective, rather than thoroughly benchmarking the services in order to optimize their workflow or implementation. Other works present a comparison of different stateful Web Services implementations [6], and analyze the service times and maximum concurrency of several Globus Grid services through an automated distributed performance testing tool [7].

In the context of Computational Grids, we can mention the following meta-scheduling projects: Condor/G [8], which provides user tools with fault tolerance capabilities to submit jobs to a Globus based Grid; Nimrod/G [9], designed specifically for Parameter Sweep Application (PSA) optimizing user-supplied parameters like deadline or budget; GridLab Resource Management System (GRMS) [10], which is a meta-scheduler component to deploy resource management systems for large scale infrastructures; and the Community Scheduler Framework (CSF) [11], an implementation of an OGSA-based meta-scheduler; and finally the Enabling Grids for E-sciencE (EGEE) Resource Broker [12], that handles job submission and accounting. On the other hand, Grid*W*ay gives end users, application developers and managers of Globus infrastructures a scheduling functionality similar to that found on local DRM systems, including the support for DRMAA GGF standard. A comparison of different approaches to Grid resource management systems can be found in [13,14].

Even though some of the aforementioned application schedulers, like Condor/G or Nimrod/G, have recently provided support for Globus WS services, we would like to remark on the advantages of the Grid*W*ay architecture in terms of flexibility, extensibility, usability and deployability. In fact, it has been successfully used to simultaneously interface to LCG middleware and Globus WS and pre-WS components.

The aim of this paper is threefold: first, to present the loosely-coupled architecture of the Grid*W*ay meta-scheduler for interfacing simultaneously with different Grid resource management services; second, to evaluate the coordinated harnessing performed by Grid*W*ay; and third, to compare the performance at user level of the pre-WS and WS GRAM services provided by the Globus Toolkit version 4.0. The ability of Grid*W*ay to simultaneously use different Grid Services eases transition to the Web Service implementation of the Globus components.

The rest of the paper is organized as follows: Section 2 introduces the Globus approach for resource management, through its GRAM component, while Section 3 introduces the Grid*W*ay approach for job management. Section 4 discusses about Grid benchmarking. Section 5 shows the results of the coordinated harnessing of both kinds of GRAM services, while Section 6 compares their performance. Finally, Section 7 ends up with some conclusions.

2. The Globus approach for resource management

The Globus Toolkit [15] has become a *de facto* standard in Grid computing. Globus services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [16]. Resource management is maybe the most important component for computational Grids, although it could also be extended to other non-computational resources. The *Grid Resource Allocation and Management* (GRAM) [17] service is the core of the resource management pillar of the Globus Toolkit.

In pre-WS GRAM (see Fig. 1), a job is submitted through the Gatekeeper service of the remote computer. The Gatekeeper is a service running on every node of a Globus Grid. The Gatekeeper handles each request, mutually authenticating with the client and mapping the request to a local user, and creating a Job Manager for each job. The Job Manager starts, controls and monitors the job according to its RSL (Resource Specification Language) specification, 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.

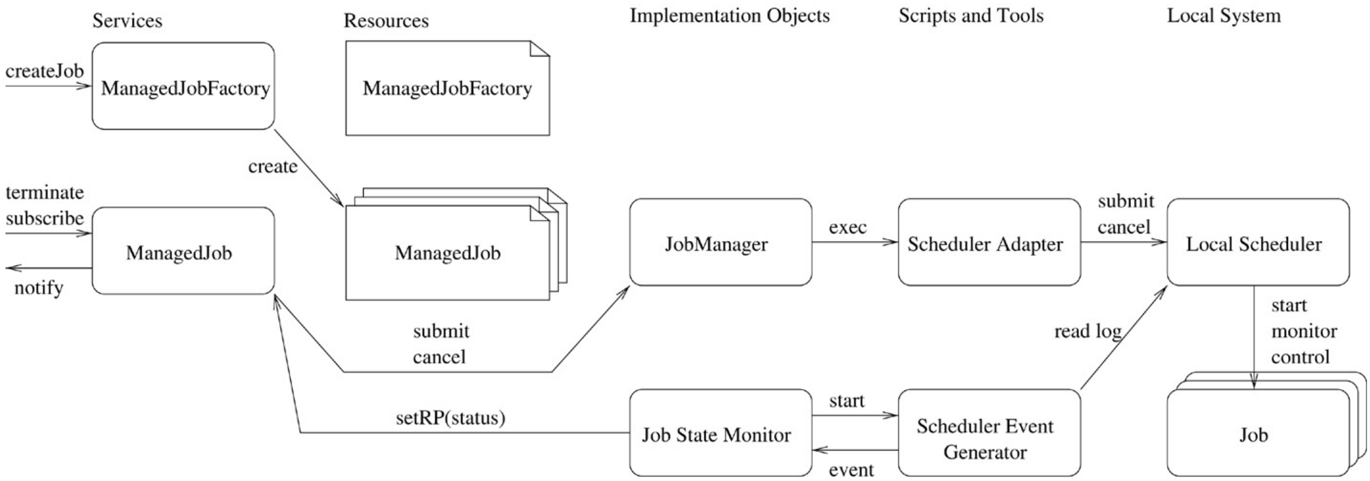


Fig. 2. Architecture of the WS *Grid Resource Allocation and Management* (GRAM) service.

In WS GRAM (see Fig. 2), when a job is submitted, the request is sent to the Managed Job Factory service of the remote computer. The Managed Job Factory and Managed Job are two services running on every node of a Globus Grid. The Managed Job Factory handles each request and creates a Managed Job resource for each job. Authentication is performed via Web Service mechanisms and some operations are mapped to a local user via the `sudo` UNIX command. The Managed Job service uses a Job Manager to start and control the job according to its RSL specification, mapping the request to a local user and communicating state changes back to the GRAM client via WS-Notifications [18]. When the job terminates, either normally or by failing, the Managed Job resource is destroyed, ending the life cycle of the Grid job.

Note also that, in pre-WS GRAM, there is one Gatekeeper which starts multiple Job Managers, while in WS GRAM, there is one Managed Job Factory service and only one Managed Job service which controls multiple jobs, acting as Managed Job resources in WSRF.

Although the use of Web Services entails some overhead, the implementation of WS GRAM has been optimized in several ways. For example, it provides better job status monitoring mechanism through the use of a *Job State Monitor* (JSM), which in turns uses a *Scheduler Event Generator* (SEG), instead of implementing a polling mechanism in the Job Manager, as in pre-WS GRAM. It also provides a more scalable/reliable file handling through the use of a *Reliable File Transfer* (RFT) service instead of the `globus-url-copy` command used directly by the Job Manager in pre-WS GRAM. Moreover, WS GRAM only supports GridFTP for file transfer and the use of the non-scalable GASS (Global Access to Secondary Storage) caching mechanism has been removed, although it is of great use for parametric jobs [19]. In any case, WSRF-based Grid services in GT4 clearly outperform heavy-weight OGSI-based Grid services in GT3 [7].

As can be seen in Fig. 2, WSRF separates services, resources and implementation objects. This way is easier to standardize a service architecture, like OGSA, since only services and resource properties representing resource state have to be specified in the standardization documents.

GRAM operates in conjunction with a number of schedulers including Condor, PBS (Portable Batch System) and a simple “fork” scheduler. The Job Manager provides a plugin architecture for extensibility. When the Job Manager is respectively invoked by the Gatekeeper or Managed Job service to process a job request, it maps the request to a local scheduler. These plugins provide a set of programs and scripts that map job requests to scheduler commands such as submit, poll or cancel.

3. The Grid*W*ay approach for job management

Grid*W*ay[1] is an open source meta-scheduling technology that provides a decentralized, modular and “end-to-end” architecture for resource brokering and job management, in dynamic and *loosely-coupled* Grid environments [20,21]. The core of the framework is a personal submission agent that performs all submission stages [16] and watches over the efficient execution of the job. Adaptation to changing conditions is achieved by dynamic rescheduling. Once the job is initially allocated, it is rescheduled when performance slowdown or remote failure are detected, and periodically at each discovering interval. Application performance is evaluated periodically at each monitoring interval.

The submission agent consists of the following components (see Fig. 3):

- *Request Manager*: To handle client requests.
- *Dispatch Manager*: To perform job scheduling.
- *Submission Manager*: To perform the stages of job execution, including job migration.
- *Execution Manager*: To execute each job stage.
- *Performance Monitor*: To evaluate the job performance.

The flexibility of the framework is guaranteed by a well-defined API (Application Programming Interface) for each submission agent component. Moreover, the framework has

[1] http://www.gridway.org.

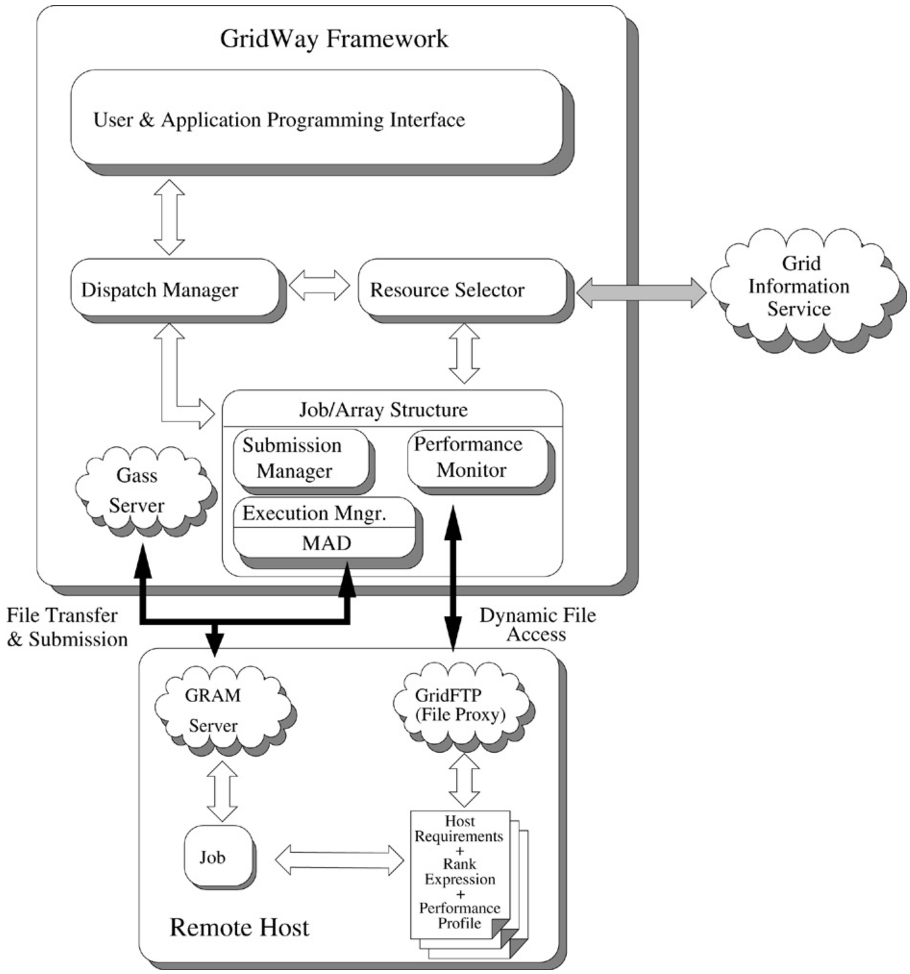


Fig. 3. Architecture of Grid*W*ay.

been designed to be modular to allow adaptability, extensibility and improvement of its capabilities. The following modules can be set on a per job basis:

- *Resource Selector*: Used by the Dispatch Manager to select the most adequate host to run each job according to the host's rank, architecture and other parameters.
- *Middleware Access Driver*: Used by the Execution Manager to submit, monitor and control each job stage.
- *Performance Evaluator*: Used by the Performance Monitor to check the progress of the job.
- *Prolog*: Used by the Submission Manager to prepare the remote machine and transfer the executable, input and restart (in the case of migration) files.
- *Wrapper*: Used by the Submission Manager to run the executable file and capture its exit code.
- *Epilog*: Used by the Submission Manager to transfer back output or restart (in case of stop) files and clean up the remote machine.

This way, the Resource Selector interfaces with Grid Information services (e.g. Globus Monitoring and Discovery Service, MDS), the Middleware Access Driver interfaces with Resource Management services (e.g. Globus GRAM), Prolog and Epilog interface with Data Management services (e.g. Globus GridFTP, Reliable File Transfer, RFT, and Data Replication Service, DRS), Wrapper interfaces with Execution services and Performance Evaluator interfaces with Performance services. The result is that the Grid*W*ay core is independent of the underlying middleware implementation.

3.1. The request manager and dispatch manager

The client application uses the *Distributed Resource Management Application API* (DRMAA) [22] to communicate with the Request Manager in order to submit the job along with its configuration file, or job template, which contains all the necessary parameters for its execution. Once submitted, the client may also request control operations to the request manager, such as job stop/resume, kill or reschedule.

The Dispatch Manager periodically wakes up at each scheduling interval, and tries to submit pending and rescheduled jobs to Grid resources. It invokes the execution of the Resource Selector module, which returns a prioritized list of candidate hosts. The Dispatch Manager submits pending jobs by invoking a Submission Manager, and also decides if the migration of rescheduled jobs is worthwhile or not. This decision can be based on the reason of rescheduling, the elapsed time, the estimated remaining time, or the estimated transfer time of input and checkpoint files [23]. If this is the case, the Dispatch Manager triggers a migration event along with the new selected resource to the Submission Manager, which manages the job migration.

3.2. The submission manager and performance monitor

The Submission Manager is responsible for the execution of the job during its lifetime, i.e. until it is done or stopped. It is invoked by the Dispatch Manager along with a selected host to submit a job, and is also responsible for performing job migration to a new resource. The Globus management components and protocols are used to support all these actions.

The Submission Manager performs the following tasks:

- Preparation: Submitting the Prolog executable, monitoring its correct execution and updating the submission states.
- Execution: Submitting the Wrapper executable, monitoring its correct execution, updating the submission states and waiting for events from the Dispatch Manager.
- Cancellation: Cancelling the submitted job if a migration, stop or kill event is received by the Submission Manager.
- Finalization: Submitting the Epilog executable, monitoring its correct execution and updating the submission states.

Therefore, Grid*W*ay doesn't rely on the underlying middleware to perform preparation and finalization tasks. Moreover, since both Prolog and Epilog are submitted to the front-end node of a cluster and Wrapper is submitted to a compute node, Grid*W*ay doesn't require any middleware installation nor network connectivity in the compute nodes. This is one of the main advantages of the "end-to-end" architecture of Grid*W*ay.

The Performance Monitor periodically wakes up at each monitoring interval. It requests rescheduling actions to detect better resources when performance slowdown is detected and at each discovering interval.

3.3. The execution manager

In order to provide an abstraction with the resource management middleware layer, the Execution Manager uses a *Middleware Access Driver* (MAD) module to submit, control and monitor the execution of the Prolog, Wrapper and Epilog modules. The MAD module provides basic operations with the resource management middleware. The use of standard input/output makes easy the debugging process of new MADs.

The format to send a request to the MAD, through its standard input, is:

`OPERATION JID HOST[/JM] RSL`

where `OPERATION` can be one of the following:

- `INIT`: Initializes the MAD.
- `SUBMIT`: Submits a job.
- `POLL`: Polls a job to obtain its state.
- `CANCEL`: Cancels a job.
- `FINALIZE`: Finalizes the MAD.

`JID` is a job identifier, chosen by Grid*W*ay, `HOST` and the optional `JM` specifies, respectively, the resource contact and job manager to submit the job if the operation is SUBMIT (otherwise they are ignored) and `RSL` specifies the resource specification to submit the job if the operation is `SUBMIT` (otherwise it is ignored).

On the other side, the format to receive a response from the MAD, through its standard output, is:

`OPERATION JID RESULT INFO`

where `OPERATION` is the operation specified in the request that originated the response or `CALLBACK`, in the case of an asynchronous notification of a state change, `JID` is the job identifier, as provided in the submission request, `RESULT` is the result of the operation (it could be `SUCCESS` or `FAILURE`) and `INFO` contains the cause of failure if `RESULT` is `FAILURE`, or it contains the state of the job, if `OPERATION` is `POLL` or `CALLBACK`.

Currently, the are two MADs available. One, written in C, interfaces with pre-WS GRAM services and other, written in Java, interfaces with WS GRAM services. *Java Virtual Machine* (JVM) initialization time doesn't affect, since the JVM is initiated before the start of measurements.

4. Benchmarks for Grid computing

Benchmarks are designed to provide an objective measure of the capabilities of hardware and software systems to execute a typical application profile. As is well known, there is no better benchmark than the own application or application set for which the Grid infrastructure has been developed for. However, it is convenient to count on well-defined test programs, since they allow the evaluation of different infrastructures by executing the same workload.

The *Grid Benchmarking Research Group* (GBRG),[2] within the *Global Grid Forum* (GGF),[3] proposes to create a set of representative Grid benchmarks [24], which will embody challenging usage scenarios with special emphasis on large data usage. The *NAS Grid Benchmarks* (NGB) [25] suite has been the first Grid benchmark specification available. It defines a set of data flow graphs that model applications typically executed on the Grid. The NGB specification suggests using the job turnaround time as basic quantitative performance metric. However, metrics like the resource usage or data transfer times between tasks are identified as useful for diagnostic purposes. Other qualitative metrics, like security and fault tolerance, are considered crucial for a successful Grid infrastructure [26]. The whole NGB suite has been previously implemented by using the DRMAA interface supported by Grid*W*ay [27].

For the experiments below, we have chosen the ED (Embarrassingly Distributed) benchmark from the NGB suite. The ED benchmark represents an important class of Grid applications called *Parameter Sweep Applications* (PSA), which constitute multiple independent runs of the same

[2] http://www.nas.nasa.gov/GGF/Benchmarks.

[3] http://www.ggf.org.

Table 1
Characteristics of the pre-WS and WS GRAM resources in the research testbed

Name	Site	Location	Nodes	Processor	Speed	Memory per node (MB)	DRMS
cygnus	UCM	Madrid	1	Intel P4	2.5 GHz	512	–
ursa	UCM	Madrid	1	Intel P4	3.2 GHz	512	fork
draco	UCM	Madrid	1	Intel P4	3.2 GHz	512	fork
hydrus	UCM	Madrid	4	Intel P4	3.2 GHz	512	PBS
aquila	UCM	Madrid	2	Intel PIII	600 MHz	250	SGE

Table 2
Characteristics of the pre-WS GRAM resources in the production testbed

Name	Site	Location	Nodes	Processor	Speed (GHz)	Memory per node	DRMS
egeece	IFCA	Cantabria	28	2 × Intel PIII	1.2	512 MB	PBS
lcg2ce	IFIC	Valencia	117	AMD Athlon	1.2	512 MB	PBS
lcg-ce	CESGA	Galicia	72	Intel P4	2.5	1 GB	PBS
ce00	INTA-CAB	Madrid	4	Intel P4	2.8	512 MB	PBS
ce01	PIC	Cataluña	65	Intel P4	3.4	512 MB	PBS

program, but with different input parameters. In this case, each task consists in the execution of the SP (Scalar Pentadiagonal) flow solver [28] with a different initialization parameter for the flow field. This kind of computations appears in many scientific fields like Biology [29], Pharmacy, or Computational Fluid Dynamics. In spite of the relatively simple structure of this application profile, its efficient execution on computational Grids involves challenging issues [19].

NGB defines several problem sizes (in terms of mesh size, iterations and number of tasks) as classes S, W, A, B, C, D and E. We have used a problem size of class A, since it is appropriate for middle-class resources. However, instead of submitting 9 tasks, as NGB class A defines, we have submitted much more tasks in order to have a real high-throughput application.

The characteristics of the ED benchmark, like high number of repeated submissions, relatively easy scheduling, and low input/output requirements, makes it very appropriate to evaluate resource management services. However, this choice doesn't affect the generality of measurements nor observations. In fact, other benchmarks in the suite (like VP or MB, from the NGB suite), made of dependent tasks, are undoubtedly better to analyze the scheduling and data movement capabilities of a scheduler [30].

5. Coordinated harnessing of pre-WS and WS GRAM services

In this section, we analyze the coordinated use of a research testbed (described in Table 1) with WS GRAM as part of Globus Toolkit 4.0, and a production testbed (described in Table 2), which is composed of some Spanish sites enrolled in EGEE[4] (Enabling Grids for E-sciencE)), with pre-WS GRAM as part of the LCG (LHC Computing Grid) middleware. Testbed resources are interconnected by the *Spanish National Research and Education Network* (RedIRIS, see Fig. 4) and several regional networks, like the *Telematic Research Network of Madrid* (REDIMadrid, see Fig. 5), which is based on DWDM (Dense Wavelength Division Multiplexing) optical technology and connects several research centers in the community of Madrid, including UCM and INTA-CAB, at 1 Gbps each. The resulting environment is highly dynamic and heterogeneous due to the shared use of compute and network resources, the different DRMS (Distributed Resource Management Systems), processors and network links, the different middleware and service technologies, etc.

The version of the Globus toolkit included in LCG has been adapted in several ways, mainly: an automatic generation of Grid map files, a new GLUE (Grid Laboratory Uniform Environment) schema [31] for MDS (Monitoring and Discovery Service), a persistent BDII (Berkeley Database Information Index) instead of GIIS (Grid Index Information Service), and the fact that file systems are not shared by default between cluster nodes. In a previous work [32], we have described the coordinated use of two Grid infrastructures, one based on Globus pre-WS services and another based on the LCG middleware, by only using the Globus pre-WS protocols and interfaces. In this work, we have extended the modularity of the Grid*W*ay framework to the resource management interfacing layer, through the MAD, in order to support the simultaneous use of both pre-WS and WS Grid services.

Scheduling is based on job requirements, resource ranks and resource availability. We have used a simple Resource Selector, consisting of a list of resources, along with their characteristics (including the MAD that should be used to access each of them). This way, the Grid Information services does not interfere with the measurements. Moreover, in order to not saturate the production testbed with this experiment, we have imposed the limitation to use only four nodes simultaneously on each compute resource. In this case, the width of the ED benchmark has been defined to be 100 tasks.

Figs. 6 and 7 show the dynamic throughput achieved and the scheduling performed, respectively, during four experiments.

4 http://www.eu-egee.org.

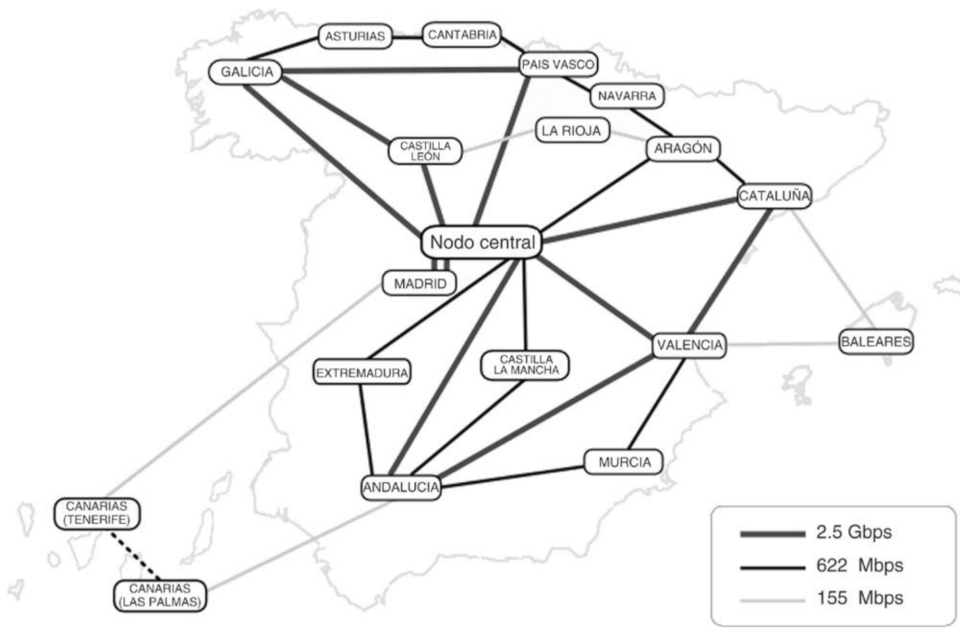


Fig. 4. Topology of the RedIRIS-2 network.

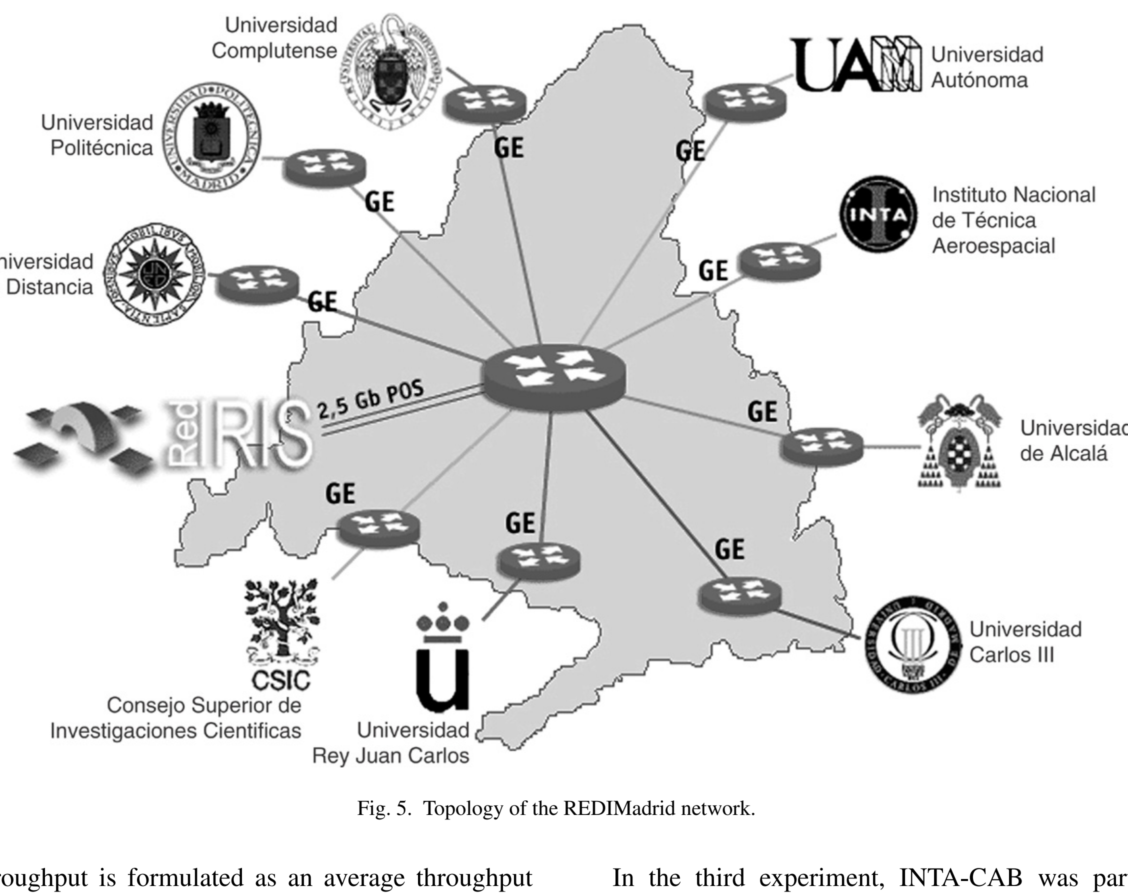


Fig. 5. Topology of the REDIMadrid network.

Dynamic throughput is formulated as an average throughput calculated every time a job completes. Experiment 1 reaches the maximum throughput (212 jobs/h) since all resources were available. During experiment 2, PIC was unavailable, so no job was allocated to this site and the other sites received more jobs. Therefore, the throughput dropped considerably (154 jobs/h).

In the third experiment, INTA-CAB was partially busy, being only two nodes available for execution. This is reflected in the schedule (INTA-CAB received half the jobs as compared to the first experiment) and in the achieved throughput (181 jobs/h). Finally, during experiment 4, CESGA and PIC received some Grid jobs not related to the experiment. In all the experiments, UCM received a higher number of jobs since

Table 3
Transfer and execution times (seconds) per job on each resource

Host	Pre-WS				WS			
	Execution time		Transfer time		Execution time		Transfer time	
	Mean	Dev.	Mean	Dev.	Mean	Dev.	Mean	Dev.
draco	225.1	0.4	22.2	0.5	229.1	4.1	31.5	6.9
ursa	205.1	0.4	22.0	0.0	215.5	2.4	31.9	4.8
hydrus	195.0	10.5	26.1	1.7	207.0	5.0	52.0	10.9
aquila	1379.0	142.8	43.0	1.4	1404.0	127.3	106.5	4.9
Total	248.8	234.3	25.5	4.3	259.8	236.8	48.0	18.4

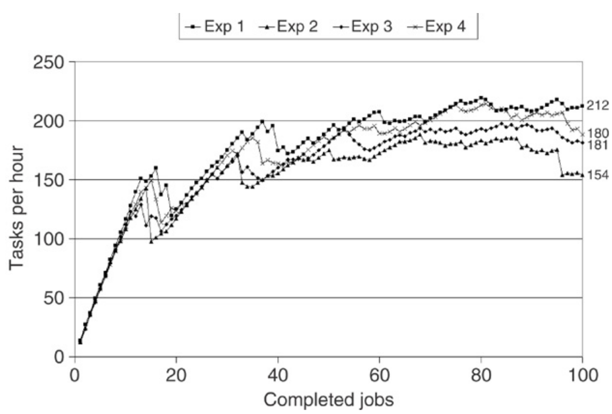


Fig. 6. Dynamic throughput in the four experiments.

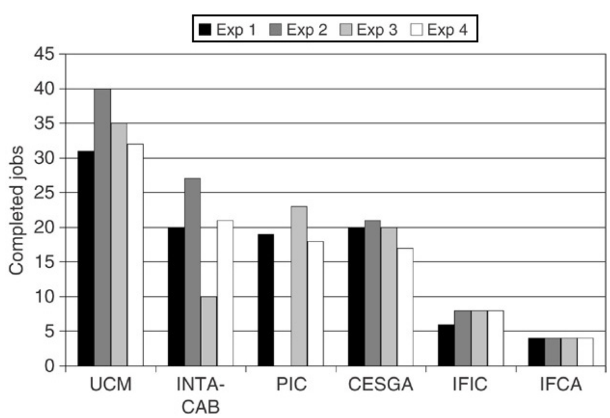


Fig. 7. Scheduling performed in the four experiments.

it presents more resources and, therefore, more compute nodes (10 vs. 4) due to the limitation of four simultaneously running jobs on the same resource.

6. Performance evaluation of pre-WS and WS GRAM services

In this section, we evaluate both implementations of Globus services. The experimental results have been obtained on the research testbed previously described in Table 1, whose resources can be accessed via either pre-WS or WS GRAM services, both from Globus Toolkit 4.0. The use of this controlled testbed allows a better comparison of results. The resources are connected through the local network of the UCM (Universidad Complutense de Madrid), which is Fast Ethernet at 100 Mbps. In the following experiments, cygnus is used as client. The ED benchmark used for the following experiments comprises the execution of 50 independent tasks.

Table 3 shows the average and deviation of the transfer (preparation and finalization stages) and execution (execution stage) times for each resource and GRAM service. Pre-WS transfer times on draco, ursa and hydrus are twice the Job Manager polling period (10 s since Globus 2.4) plus some overhead. Also, aquila presents a higher execution and transfer time since its compute nodes have slower processors. Even though, ursa, draco and hydrus's compute nodes are identical, hydrus presents a lower mean execution time, due to the exclusive access to the compute nodes provided by the DRMS, but a higher deviation, due to the overhead of the DRMS and the simultaneous submission of multiple jobs. It also presents a higher transfer time, due to the simultaneous transfer of files for multiple jobs. In the case of pre-WS services, this is alleviated through the use of the GASS cache.

Table 4 shows detailed times for all jobs obtained in two of the experiments. Suspension and active times are measured by Grid*W*ay by following the GRAM protocol, and total time is the sum of the previous two. Real time is directly measured by Prolog, Wrapper and Epilog modules as the time actually spent on their activities, and overhead time is the difference between total and real time. Table 5 again shows detailed times, but only for those jobs submitted to hydrus. Regarding the total execution time, the performance gain in pre-WS is lower than 6%. It can be seen that the suspension time (time from submission to active state) is greater in WS GRAM due to the Web Service container overheads, the use of credential delegation and file transfer WS Grid services and the lack of GASS caching. This results in overhead times that are roughly twice in WS GRAM. Moreover, the time actually spent (i.e. the real time) on preparation and finalization stages is also greater, which could be due also to container overheads.

Fig. 8 shows the dynamic throughput achieved during the experiments. It is clear that using pre-WS a higher throughput is reached (83 vs. 78 jobs/h). However, this does not suppose a big difference in performance (only 6%). Thus, in spite of the undoubtedly greater overheads seen in WS GRAM, this is not appreciated in the achieved throughput, since the ED benchmark takes much more time to execute than the preparation and finalization stages.

Table 4
Detailed times (seconds) per job

Time	Pre-WS			WS		
	Prolog	Wrapper	Epilog	Prolog	Wrapper	Epilog
Suspension	2.9	6.2	1.9	12.3	9.9	9.5
Active	10.4	242.6	10.0	14.4	247.6	13.2
Total	13.3	248.8	11.9	26.7	257.5	22.7
Real	0.6	238.5	0.5	2.3	237.4	1.6
Overhead	12.7	10.3	11.4	24.4	20.1	21.1

Table 5
Detailed times (seconds) per job on hydrus

Time	Pre-WS			WS		
	Prolog	Wrapper	Epilog	Prolog	Wrapper	Epilog
Suspension	3.0	6.4	2.2	13.1	9.7	10.5
Active	10.4	188.6	10.4	14.5	196.6	12.3
Total	13.4	195.0	12.6	27.6	206.3	22.8
Real	0.6	184.5	0.7	2.3	186.1	1.8
Overhead	12.8	10.5	11.9	25.3	20.2	21.0

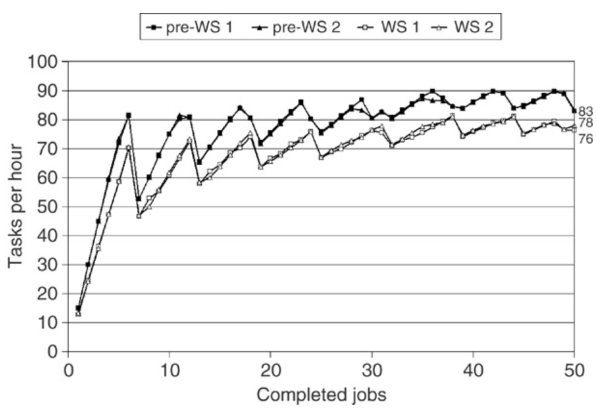


Fig. 8. Dynamic throughput in the four experiments.

The scheduling performed during the four experiments is identical in all the experiments, allocating 32 jobs to hydrus, 8 jobs to draco and ursa and 2 jobs to aquila.

7. Conclusions

The Grid*W*ay meta-scheduler is able to work over different infrastructures in a *loosely-coupled* way, allowing a straightforward resource sharing. The smooth process of integration of two so different infrastructures and service technologies demonstrates that the Grid*W*ay approach, based on a modular, decentralized and "end-to-end" architecture, is appropriate for the Grid. The proposed modular architecture for job management eases the gradual migration from pre-WS Grid services to WS ones, and even, the long-term coexistence of both.

The experimental results demonstrate that WS-based GRAM has more overheads compared to pre-WS GRAM. However, for high-throughput applications that does not pose a big issue. On the other hand, the Web Service interface for GRAM provides additional benefits, like superior scalability, partly thanks to its improved implementation. Moreover, it is expected that a new implementation of GRAM over the C WS Core (currently it is implemented over the Java WS Core) will reduce this overhead and improve performance. More detailed results show that the WS GRAM implementation would benefit from a mechanism for file caching, mainly for parametric jobs, implemented in the RFT service.

Acknowledgements

We would like to acknowledge all the institutions that have contributed resources to perform the experiments. They are *Departamento de Arquitectura de Computadores y Automática* (DACyA) at *Universidad Complutense de Madrid* (UCM), *Centro de Astrobiología* (CAB) at *Instituto Nacional de Técnica Aeroespacial "Esteban Terradas"* (INTA), *Centro de Supercomputación de Galicia* (CESGA), *Instituto de Física de Cantabria* (IFCA), *Instituto de Física Corpuscular* (IFIC) and *Port d'Informació Científica* (PIC).

References

[1] I. Foster, What is the Grid? A three point checklist, GRIDtoday 1 (6), Available from http://www.gridtoday.com/02/0722/100136.html.

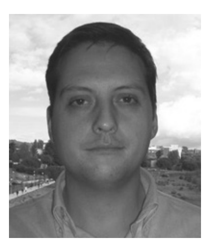
[2] K. Czajkowski, D.F. Ferguson, I. Foster et al., The WS-Resource Framework Version 1.0, Tech. Rep. Available from http://www.globus.org/wsrf/specs/ws-wsrf.pdf (2004).

[3] I. Foster, K. Czajkowski, D.E. Ferguson, et al., Modeling and managing state in distributed systems: The role of OGSI and WSRF, Proceedings of the IEEE 93 (3) (2005) 604–612.

[4] S. Tuecke, K. Czajkowski, I. Foster et al., Open Grid services infrastructure (OGSI) version 1.0, Tech. Rep. GFD-R-P.15, Open Grid Services Infrastructure Working Group—The Global Grid Forum, 2004.

[5] I. Foster, C. Kesselman, J. Nick, S. Tuecke, The Physiology of the Grid: An open Grid services architecture for distributed systems integration, Tech. Rep., Open Grid Service Infrastructure Working Group—The Global Grid Forum, 2002.
[6] M. Humphrey, G. Wasson, K. Jackson et al., State and events for web services: A comparison of five WS-Resource framework and WS-notification implementations, in: 14th IEEE Intl. Symp. High Performance Distributed Computing, HPDC-14, 2005, pp. 3–13.
[7] I. Raicu, A performance study of the globus toolkit and Grid services via DiPerF, an Automated Distributed Performance testing framework, Master's Thesis, University of Chicago, Computer Science Department, 2005.
[8] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A computation management agent for multi-institutional Grids, Journal of Cluster Computing 5 (3) (2002) 237–246.
[9] R. Buyya, D. Abramson, J. Giddy, A computational economy for Grid computing and its implementation in the Nimrod-G Resource broker, Future Generation Computer Systems 18 (2002) 1061–1074.
[10] E. Seidel, G. Allen, A. Merzky, J. Nabrzyski, GridLab—A Grid application toolkit and testbed, Future Generation Computer Systems 18 (8) (2002) 1143–1153.
[11] Open source metascheduling for virtual organizations with the community scheduler framework (CSF), Tech. Rep., Platform Computing, August 2003.
[12] EGEE middleware architecture and planning (Release 2), Tech. Rep. DJRA1.4, EGEE, July 2005.
[13] Y. Gaoa, H. Rongb, J.Z. Huangc, Adaptive Grid job scheduling with genetic algorithms, Future Generation Computer Systems 21 (2005) 151–161.
[14] K. Krauter, R. Buyya, M. Maheswaran, A taxonomy and survey of Grid resource management systems for distributed computing, Software—Practice and Experience 32 (2) (2002) 135–164.
[15] I. Foster, C. Kesselman, The globus project: A status report, Future Generation Computer Systems 15 (1999) 607–621.
[16] J.M. Schopf, Ten actions when superscheduling, Tech. Rep. GFD-I.4, Scheduling Working Group—The Global Grid Forum, 2001.
[17] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, A resource management architecture for metacomputing systems, in: Proc. IPPS/SPDP'98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998, pp. 62–82.
[18] S. Graham, P. Niblett et al., Publish-subscribe notification for web services version 1.0, Tech. Rep., 2004. Available from http://www.globus.org/wsrf.
[19] E. Huedo, R.S. Montero, I.M. Llorente, Experiences on adaptive Grid scheduling of parameter sweep applications, in: Proc. 12th Euromicro Conf. Parallel, Distributed and Network-based Processing, PDP 2004, IEEE CS, 2004, pp. 28–33.
[20] E. Huedo, R.S. Montero, I.M. Llorente, A framework for adaptive execution on Grids, Software—Practice and Experience 34 (7) (2004) 631–651.
[21] I.M. Llorente, R.S. Montero, E. Huedo, A loosely-coupled vision for computational Grids, IEEE Distributed Systems Online 6 (5).
[22] H. Rajic, R. Brobst, W. Chan et al., Distributed resource management application API specification 1.0, Tech. Rep. GFD-R-P.22, DRMAA Working Group—The Global Grid Forum (2003).
[23] R.S. Montero, E. Huedo, I.M. Llorente, Grid resource selection for opportunistic job migration, in: Proc. 9th Intl. Conf. Parallel and Distributed Computing (Euro-Par 2003), in: LNCS, vol. 2790, 2003, pp. 366–373.
[24] A. Snavely, G. Chun, H. Casanova, R.F. Van der Wijngaart, M.A. Frumkin, Benchmarks for Grid computing: A review of ongoing efforts and future directions, ACM SIGMETRICS Performance Evaluation Review 30 (4) (2003) 27–32.
[25] R.F. Van der Wijngaart, M.A. Frumkin, NAS Grid Benchmarks Version 1.0, Tech. Rep. NAS-02-005, NASA Advanced Supercomputing (NAS) 2002.
[26] M.A. Frumkin, R.F. Van der Wijngaart, NAS Grid benchmarks: A tool for Grid space exploration, Journal of Cluster Computing 5 (3) (2002) 247–255.
[27] J. Herrera, E. Huedo, R.S. Montero, I.M. Llorente, Developing Grid-aware applications with the distributed resource management application API, in: Proc. 10th Intl. Conf. Parallel and Distributed Processing (Euro-Par 2004), in: LNCS, vol. 3149, 2004, pp. 429–435.
[28] D.H. Bailey, E. Barszcz, J.T. Barton, The NAS parallel benchmarks, Journal of Supercomputer Applications 5 (3) (1991) 63–73.
[29] E. Huedo, U. Bastolla, R.S. Montero, I.M. Llorente, A framework for protein structure prediction on the Grid, New Generation Computing 23 (4) (2005) 277–290.
[30] E. Huedo, R.S. Montero, I.M. Llorente, An evaluation methodology for computational Grids, in: Proc. 2005 Intl. Conf. High Performance Computing and Communications, HPCC 2005, in: LNCS, vol. 3726, 2005, pp. 499–504.
[31] S. Andreozzi, S. Burke, L. Field et al., GLUE Schema Specification version 1.2, Tech. Rep., 2005. Available from http://infnforge.cnaf.infn.it/glueinfomodel.
[32] J.L. Vázquez-Poletti, E. Huedo, R.S. Montero, I.M. Llorente, Coordinated harnessing of the IRISGrid and EGEE testbeds with GridWay, Journal of Parallel and Distributed Computing 66 (5) (2006) 763–771.

Eduardo Huedo received his M.E. in Computer Science (1999) and his Ph.D. in Computer Architecture (2004) from the Universidad Complutense de Madrid (UCM). He is Assistant Professor of Computer Architecture and Technology at UCM since 2006. Previously, he worked as Postdoctoral Researcher in the Advanced Computing Laboratory at Centro de Astrobiología (CSIC-INTA), associated to NASA Astrobiology Institute. His research areas are Performance Management and Tuning, Parallel and Distributed Computing and Grid Technology.



Rubén S. Montero received his B.S. in Physics (1996), M.S. in Computer Science (1998) and Ph.D. in Computer Architecture (2002) from the Universidad Complutense de Madrid (UCM). He is Associate Professor of Computer Architecture and Technology at UCM. He has held several research appointments at ICASE (NASA Langley Research Center), where he worked on computational fluid dynamics, parallel multigrid algorithms and Cluster computing. Nowadays, his research interests lie mainly in Grid Technology, in particular in adaptive scheduling, adaptive execution and distributed algorithms.

Ignacio M. Llorente received his B.S. in Physics (1990), M.S. in Computer Science (1992) and Ph.D. in Computer Architecture (1995) from the Universidad Complutense de Madrid (UCM). He is Executive M.B.A. by Instituto de Empresa since 2003. He is Professor of Computer Architecture and Technology in the Department of Computer Architecture and System Engineering at UCM and Senior Scientist at Centro de Astrobiología (CSIC-INTA), associated to NASA Astrobiology Institute. He has held several appointments since 1997 as a Consultant in High Performance Computing and Applied Mathematics at ICASE (NASA Langley Research Center). His research areas are Information Security, High Performance Computing and Grid Technology.