CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2010; **00**:1–14
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

Interoperating Grid Infrastructures with the GridWay Metascheduler

Ismael Marín Carrión*, Eduardo Huedo, Ignacio M. Llorente

{*i.marin, ehuedo*}*@fdi.ucm.es, llorente@dacya.ucm.es*
Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, Spain

SUMMARY

This paper describes the GridWay Metascheduler and exposes its latest and future developments, mainly related to interoperability and interoperation. GridWay enables large-scale, reliable and efficient sharing of computing resources over grid middleware. To favor interoperability, it shows a modular architecture based on drivers, which access middleware services for resource discovery and monitoring, job execution and management, and file transfer. This paper presents two new execution drivers for BES and CREAM services, and introduces a remote BES interface for GridWay. This interface allows users to access GridWay's job metascheduling capabilities, using the BES implementation of GridSAM. Thus, GridWay now provides to end-users more possibilities of interoperability and interoperation. Copyright © 2010 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: GridWay; Metascheduling; Interoperation; Grid

1. INTRODUCTION

The GridWay metascheduler [1] enables large-scale, reliable and efficient sharing of computing resources over different grid middlewares, such as Globus Toolkit [2, 3] or gLite [4]. Computing resources can be managed by different Local Resource Management Systems (LRMS) within a single organization or scattered across several administrative domains. GridWay provides a single point of access to the computing resources, from in-house systems to partner grid infrastructures and public Cloud providers. GridWay has been adopted by several research project and computational infrastructures, and has been successfully applied on different applications from science and engineering, such as life-sciences [5], aerospace [6], fusion physics [7] or computational chemistry [8].

There are a number of grid infrastructures and middlewares that have been developed over the last years. The interaction between these technologies is still a challenging problem, because they are not interoperable. Therefore, interoperation techniques are needed to allow Virtual Organizations (VO) to access the resources provided by different grids. By grid interoperability we refer the ability of grid technologies to interact directly via common open standards, while grid interoperation is a short term achievement to get grids to work together fast [9]. Interoperation techniques include the use of adapters, translators and gateways.

*Correspondence to: Departamento de Arquitectura de Computadores y Automática, Facultad de Informática, Universidad Complutense de Madrid, Spain

GridWay provides support for some of the few established standards for interoperability, like Distributed Resource Management Application API (DRMAA) [10] [11], Job Submission Description Language (JSDL) [12] or Web Services Resource Framework (WSRF) [13], but in the meanwhile, it also provides components to allow the interoperation of different grid technologies [14], like drivers, acting as adapters for different grid services, and the GridGateWay, which is a WSRF Globus Resource Allocation Manager (GRAM) service encapsulating an instance of GridWay, thus providing a gateway to resource management services [15].

Moreover, the driver-based architecture of GridWay can provide access even to heterogeneous non-grid resources. In fact, systems with high capacity computing power like HPC are not commonly accessible through any grid middleware. A prototype based on the GridWay metascheduler was proposed to access grid and non-grid computing resources in a seamless manner [16].

The purpose of this paper is to present the latest developments in GridWay concerned to achieve more interoperability and interoperation at the metascheduler level. This way, GridWay now provides support for the Basic Execution Service (BES) standard [17] both as a client and as a server, by means of a new execution driver and a new interface respectively. It also provides a new execution driver to interoperate with the Computing Resource Execution And Management (CREAM) service.

The use of adapters (like the BES and CREAM drivers) and, in particular, gateways (like the BES interface) to achieve interoperation could negatively affect performance, but the benefits of increasing the number of available resources could also overcome these drawbacks [14, 15].

Regarding related work, there are many efforts to interoperate different grid middlewares (for example, Globus and UNICORE [18]) and approaches based on portals [19] or meta-brokers [20] providing access to resources outside one grid domain, as well as interoperable metaschedulers [21]. It is also interesting to see efforts like InterGrid [22], which proposes the creation of InterGrid Gateways (IGGs) to interconnect the existing different grid islands, or GridX1 [23], which is a whole grid that could be accessed as another resource of the LHC Computing Grid (LCG). Finally, a remarkable work is being done by the Grid Interoperation Now (GIN) community group of the Open Grid Forum (OGF), which coordinates interoperation efforts in support of applications that use resources in different grids [9]. In fact, the BES interface presented in this paper was demonstrated in the last meeting of this group [24].

This paper is structured as follows. Section 2 presents the European grid scenario that explains the need for interoperability and interoperation on European Grid infrastructures, and how this work contributes to enhance these capabilities of grid infrastructures. Section 3 introduces the GridWay technology that underlies this work. Sections 4 and 5 present the two new execution drivers, CREAM and BES respectively, which provide more interoperation and interoperability to end-users. Section 6 presents a GridWay DRMAA Connector for GridSAM that enables the interoperable submission and control of jobs by means of its BES interface. Section 7 exposes the experiments performed with the new drivers. Finally, Section 8 presents the forthcoming work and sums up the main conclusions.

2. THE EUROPEAN GRID SCENARIO

The European Grid Infrastructure (EGI)[†] mission is to enable access to computing resources for European researchers from all fields of science. EGI hosts the Unified Middleware Distribution (UMD), which is a integrated set of software components offered for deployment on the EGI production infrastructure. Services included in a UMD release should be fully interoperable, where applicable through the adoption of established standards. To this end, EGI defines a set of capabilities that should be met by UMD services, including the existing standards to provide them. For example, the BES specification is recommended for the job submission

[†] http://www.egi.eu/

(Compute.JobExecution) and metascheduling (Compute.JobScheduling) capabilities. Currently, different gLite, UNICORE, Advanced Resource Connector (ARC) and Globus services are part of UMD, providing different sets of capabilities.

EGI provides an infrastructure which is accessible by means of different VO's and it is based on the federation of individual National Grid Infrastructures (NGI). For example, the Ibergrid collaboration allows the interoperation between the Spanish NGI (ES-NGI) and Portuguese NGI (PT-NGI) thanks to an agreement between Spain and Portugal. This initiative manages the deployment of distributed services on both NGIs in the framework of EGI. The Ibergrid infrastructure will be used later in the experiments of Section 7.1 Among other grid services, the EGI infrastructure provides information services, such as the Berkeley Database Information Index (BDII), execution services (CREAM and GRAM) and data transfer services (GridFTP).

The Initiative for Globus in Europe (IGE)[‡] project collaborates actively with EGI, for example, providing new technology components to EGI's UMD. IGE is a project funded by the European Union's 7th Framework Programme to coordinate European Globus activities, and it is currently supporting the development of the GridWay metascheduler. This component is scheduled for inclusion in the next UMD releases. IGE also provides a testbed composed of Globus services distributed across multiple organizations. The IGE testbed has a GridWay instance acting as a single point of access to the IGE computing resources. This testbed will be also used later in the experiments of Section 7.2.

This work is motivated by the need of interoperation and interoperability on grid environments like those presented above. In particular, this paper presents novel developments that enable the interoperation at the metascheduler layer. These innovative proposals improve the interoperation capabilities of GridWay by means of the provision of CREAM and BES execution drivers, and a BES interface. The CREAM execution driver enables the submission of jobs to infrastructures managed by the gLite middleware, while the BES one allows GridWay to submit jobs to BES endpoints. Finally, the BES interface enables the remote access to GridWay's metascheduling capabilities through a standard interface.

3. THE GRIDWAY METASCHEDULER

GridWay consists of a modular structure formed by User Interface (UI), GridWay core, scheduler module, and Middleware Access Drivers (MAD). This modular architecture favors the interoperation with new grid services, by the aggregation of new MADs that serve as adapters to these services. Fig. 1 shows the main components of the GridWay architecture.

The UI consists of LRMS-like commands that enable the end user to submit, kill, reschedule, monitor and synchronize jobs. GridWay manages the execution of single, array and workflow jobs, and tasks can be parallel. Jobs are described according to the job templates, where job requirements are specified (executable, arguments, resource requirements, etc.). The UI also provides a DRMAA implementation to develop distributed applications [25]. This includes C, Java, Perl, Python and Ruby bindings.

The GridWay core is responsible for job execution management and resource brokering, providing advanced scheduling, and job failure and recovery capabilities. GridWay core implements a Request Manager (RM) which manages all end user requests. The Dispatch Manager (DM) manages all submission stages and watches over the efficient execution of the job. GridWay core has other three modules that, through their own MADs, interface with the available services in the grid infrastructure. They are the Information Manager (IM) that performs the host discovery and monitoring, the Execution Manager (EM) that is responsible for job execution management and monitoring, and the Transfer Manager (TM) that is competent for file staging, remote working directory set-up and remote host clean-up.

[‡] http://www.ige-project.eu/

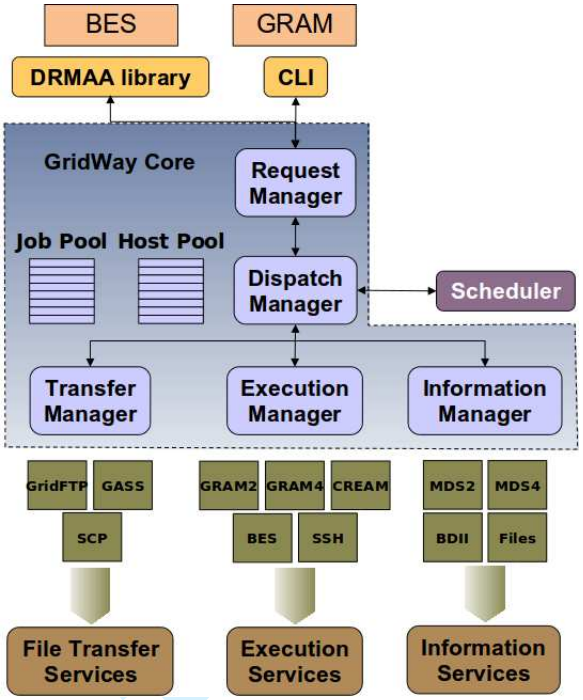

Figure 1. The GridWay architecture.

The information MADs interface with the monitoring and discovery services. GridWay supports dynamic host information data services, such as the Globus Monitoring and Discovery System (MDS) and the gLite BDII, as well as static mechanisms based on host description files. The execution MADs interface with the job management services. GridWay provides execution MADs that interface with Globus GRAM, gLite CREAM and OGF BES. Also, there is a Secure Shell (SSH) driver for non-grid resources. Finally, the transfer MADs are used to interface with the available data transfer services, such as Globus GridFTP and Global Access to Secondary Storage (GASS), as well as Secure Copy (SCP) for non-grid resources. Once there is a set of drivers implementing basic operations for a given middleware, the users benefit for the whole set of features implemented in the core, like array jobs (e.g. for parametric studies), jobs with dependencies (e.g. to create workflows), parallel jobs, advanced scheduling policies, etc.

The scheduling process is totally decoupled from the GridWay core. This process is implemented by a scheduler module that performs all scheduling decisions for jobs considering the available grid resources. This way it is possible to implement scheduling policies adapted to specific deployments without modifying the core [26]. In order to perform the scheduling decisions, the scheduler considers information coming from the list of pending jobs, the resource match-making results, the current resource behavior and the past grid usage. The information gathered from the previous sources is combined with a given scheduling policy to prioritize jobs and resources. GridWay combines job and resource prioritization policies to implement a wide range of scheduling schemes.

4. EXECUTION DRIVER FOR CREAM

The gLite CREAM service is a simple, lightweight service for job management operation at the Computing Element (CE) level, which is the gLite service representing a computing resource [27]. CREAM accepts job submission requests, described using the Job Description Language (JDL), as well as other job management and monitoring requests.

DOI: 10.1002/cpe

The new CREAM driver for GridWay provides an abstraction with the resource management layer of the gLite middleware that enables to submit, control and monitor the execution of jobs. Thus, it implements basic operations to interface with CREAM. They are:

- INIT: Initializes the driver.
- SUBMIT: Submits a job to a CREAM resource. This operation expects as arguments, a job identifier chosen by GridWay, the CREAM resource contact and the job manager to submit the job, and the path to the JDL file that describes the job.
- POLL: Queries the status of a job. This operation expects as argument the job identifier.
- RECOVER: Recovers a job, taking as input the identifier returned by the CREAM resource after the job has been successfully submitted.
- CANCEL: Cancels the job identified by the given identifier.
- FINALIZE: Finalizes the driver.

Each operation returns SUCCESS or FAILURE and other complementary information, such as the job state after a POLL operation or the CREAM job identifier after a SUBMIT operation.

The GridWay core calls these driver operations by means of the EM module, hiding the CREAM implementation details to the end user and the core. GridWay core schedules synchronous polling operations in order to track the job status and control when the job execution is finished. However, CREAM does not currently support asynchronous notifications about job state changes, so it depends on synchronous job state polling. Notifications were previously provided by the CEMonitor [28], but now it has been replaced by a new `queryEvent` operation in CREAM that returns a selected range of particular events, like job state changes. Submission operations integrate the delegation of credentials to each CE. Credential renewal is automatically carried out by the driver.

MADs are configured and selected via the GridWay configuration interface. CREAM uses JDL to describe the jobs, so GridWay job templates must be translated to JDL. This is carried out by the GridWay core, and it is absolutely transparent for the end-users.

The CREAM driver along with the BDII MAD enables the interoperation with the job management services and the monitoring and discovering services provided by the gLite middleware. The Dummy transfer MAD is used to achieve full interoperation with gLite. This driver can be configured to use a GridFTP or GASS server on the client, so that job transfers are initiated from the worker node.

5. EXECUTION DRIVER FOR OGSA-BES

The Open Grid Services Architecture (OGSA) BES specification [17] defines Web Services interfaces for creating, monitoring, and controlling computational entities such as UNIX or Windows processes, Web Services or parallel programs, called activities, within a defined environment. Clients define activities using JSDL [12]. A BES implementation executes each activity that it accepts on an appropriate computational resource, which may be a single computer, a cluster managed through a resource manager, a Web Service hosting environment, or even another BES implementation [29].

Given the importance of interoperating with BES-enabled endpoints, a BES driver for GridWay has been developed, which has been tested against the BES implementation provided by GridSAM. The new BES driver provides an abstraction layer that enables users to submit jobs to BES interfaces, and control and monitor the execution of jobs. Thus, this driver implements the same basic operations discussed in the previous section that hide the BES implementation details to the end user and the core.

As all of the other MADs, GridWay core is responsible for calling the driver functions by asynchronous requests of end users (e.g. job submitting or canceling operations), or by requests managed by the EM module (e.g. job polling). The EM is also responsible for generating a valid JSDL file given a GridWay job template describing the job requirements.

The BES driver can be used with the Dummy transfer MAD and any information MADs for performing all stages during the job life-cycle in GridWay.

6. BES INTERFACE FOR GRIDWAY: A GRIDWAY DRMAA CONNECTOR FOR GRIDSAM

This section describes the remote BES interface for GridWay. This approach follows the same model as GridGateWay [15], which represents a WSRF GRAM interface for GridWay. The BES implementation of GridSAM[§] [30] is used for this purpose. GridSAM provides a job submission interface for submitting computational jobs to many commonly used distributed resource management systems, but does not provide metascheduling capabilities. Thus, a GridWay connector for GridSAM has been developed which allows the interoperation of both systems. Fig. 2 shows the architecture of the developed gateway that enables the submission and control of jobs to GridWay by means of a BES interface.

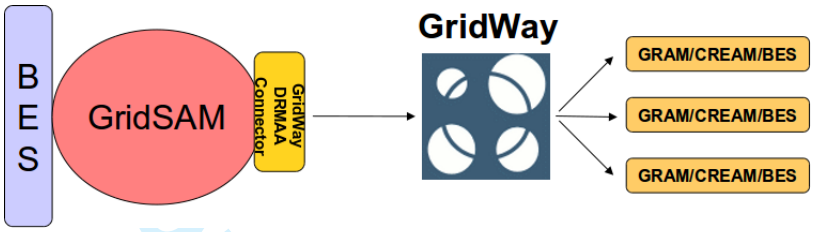

Figure 2. Architecture of the BES remote interface for GridWay.

This component enables the remote access to GridWay's metascheduling capabilities through a BES interface, allowing users to access services provided by different grid middlewares. Communication between the connector and a GridWay instance is based on the DRMAA API. The DRMAA API is an specification [10] for the submission and control of jobs to one or more LRMS. This way, the provided functionality is the same as provided by the Java DRMAA binding of GridWay, which acts as the local GridWay interface. The GridWay DRMAA Connector is responsible for handling the input JSDL document to extract all job requirements and then set up the DRMAA job attributes.

Under the end-user's point of view, jobs are submitted to a BES endpoint and are managed, monitored and executed as usual. Job attributes and requirements must be described in JSDL, according to the BES specification, which is already supported in many submission portals and tools.

The main advantage of this remote interface is that it provides a standards-based gateway to otherwise non-interoperable grid infrastructures and it lets the federation of grid resources under a single point of access.

7. EXPERIMENTS

This section presents a set of experiments for evaluating the new GridWay MADs. A first experiment is concerned with the evaluation of GRAM2 and CREAM drivers using the EGI infrastructure. Other experiments will evaluate the GRAM5 and BES drivers using the IGE testbed.

A parametric application that computes the first 10 million prime numbers is used for performing the experiments. Each experiment consists of submitting one array of 100 jobs, where each one computes a different range of 100,000 points. Thus, the job template for each job in the array is:

```
EXECUTABLE = primes.exe
ARGUMENTS = ${PARAM} 'expr ${PARAM} + 100000'
STDOUT_FILE = stdout_file.${TASK_ID}
STDERR_FILE = stderr_file.${TASK_ID}
RANK = CPU_MHZ
```

[§] http://www.omii.ac.uk/wiki/GridSAM

The range of points is calculated using the GridWay variable `PARAM`, which is calculated as `start + increment * TASK_ID`, where the values for `start` and `increment` are specified by the user when submitting the array job (in this case, 1 for `start` and 100,000 for `increment`), while `TASK_ID` is the task identifier within the job array. Both variables are substituted at run time with its corresponding value. Each job in the array will provide a partial output showing the prime numbers between the two given integers. Finally, the job template uses the rank expression to prioritize resources with higher CPU speed. Those candidates with higher rank are used first to execute the jobs. The rank is a numerical expression evaluated for each candidate host, and so those candidates with higher ranks are used first to execute the jobs.

7.1. Using GRAM and CREAM resources in Ibergrid

For the first test, a GridWay instance has been configured to access to the computing resources provided by the Information and Communication Technology (ICT) VO of Ibergrid, which is part of the EGI infrastructure, as explained before. Next, the specific configuration of the MADs for the Ibergrid ICT VO is explained.

```
IM_MAD = bdii_cream:gw_im_mad_bdii:-s bdii-egee.bifi.unizar.es
         -q (GlueCEAccessControlBaseRule=VO\:ict.vo.ibergrid.eu)
            (GlueCEImplementationName=CREAM):dummy:cream
IM_MAD = bdii_gram2:gw_im_mad_bdii:-s bdii-egee.bifi.unizar.es
         -q (GlueCEAccessControlBaseRule=VO\:ict.vo.ibergrid.eu)
            (GlueCEImplementationName=LCG-CE):dummy:gram2
EM_MAD = gram2:gw_em_mad_gram2::rsl_nsh
EM_MAD = cream:gw_em_mad_cream::jdl
TM_MAD = dummy:gw_tm_mad_dummy:-g
```

Two BDII MADs (identified by `bdii_cream` and `bdii_gram2`) that interfaces with a BDII server provided by the EGI infrastructure have been configured. They filter the computing resources by VO and implementation of the job management interface (CREAM and GRAM). Thus, the GridWay instance only use those resources registered in the VO. Each information MAD associates an execution MAD and a transfer MAD to access every discovered host. In particular, the first information MAD associates a Dummy transfer MAD and a CREAM MAD (identified by `dummy` and `cream`), while the second associates the same transfer MAD and a GRAM2 MAD (identified by `dummy` and `gram2`).

The CREAM MAD enables the interoperation with the CREAM services provided by the grid, using the `jdl` generation function to describe the jobs. The GRAM2 MAD enables the interoperation with the pre-WS GRAM services available in the infrastructure. This driver uses the `rsl_nsh` generation function to describe job requests according to the Resource Specification Language (RSL) specification, and it is used for resources with non-shared home directories.

Finally, the Dummy MAD, configured to use GASS, is employed to perform the data staging between local and remote hosts. A GASS server is started on the client and the transfer is initiated on the remote system using a Globus transfer client. Therefore, data transfers are performed through a reverse server model. This driver is intended to be used with those resources which do not have a shared home.

Table I shows the CREAM resources available for this test, and Table II shows the GRAM2 resources available.

The experiment has been carried out under some limitations in order to avoid stressing the resources and to spread the submission of jobs among resources. The job policies let only 30 simultaneously submitted jobs per user, and 10 submitted jobs per resource. Jobs are preferentially submitted to resources providing faster CPUs, as expressed in the GridWay job template.

DOI: 10.1002/cpe

Table I. CREAM resources available in the ICT VO of Ibergrid.

HOSTNAME	ORGANIZATION	LOCATION	MHZ	Nodes	LRMS
grid001	UPORTO	Porto	2400	22	SGE
ce03	IFIC	Valencia	3000	848	PBS
ce	IAA	Granada	3200	512	PBS
gridce01	IFCA	Santander	2000	2320	SGE
ce02	IFIC	Valencia	3000	368	PBS
nuredduna	UIB	Palma de Mallorca	2700	480	SGE
cream01	BIFI	Zaragoza	2667	420	PBS
grid-glite-ce-00	SGAI-CSIC	Madrid	2000	1655	PBS
creamc	BIFI	Zaragoza	2667	420	PBS
ngiescream	I3M-UPV	Valencia	2400	102	PBS

Table II. GRAM2 resources available in the ICT VO of Ibergrid.

HOSTNAME	ORGANIZATION	LOCATION	MHZ	Nodes	LRMS
ce-sge-ngi	CETA-CIEMAT	Trujillo	2000	112	SGE
ce01	UPORTO	Porto	2400	34	SGE
grid001	UPORTO	Porto	2400	22	SGE
egeece03	IFCA	Santander	2334	2320	SGE
ce01	UNICAN	Santander	2400	530	PBS
ce01	ESAC-ESA	Madrid	3200	28	SGE
ce01	CIEMAT	Madrid	3200	284	PBS
ce04	INGRID	Lisbon	2300	16	SGE
ce2	CESGA	Santiago de Compostela	2200	500	SGE

Figures 3 and 4 show the results obtained. Fig. 3 shows the total number of jobs submitted to each host, presenting how many jobs were successfully completed or had to be rescheduled due to different causes. Fig. 4 shows the evolution of the completed jobs over time.

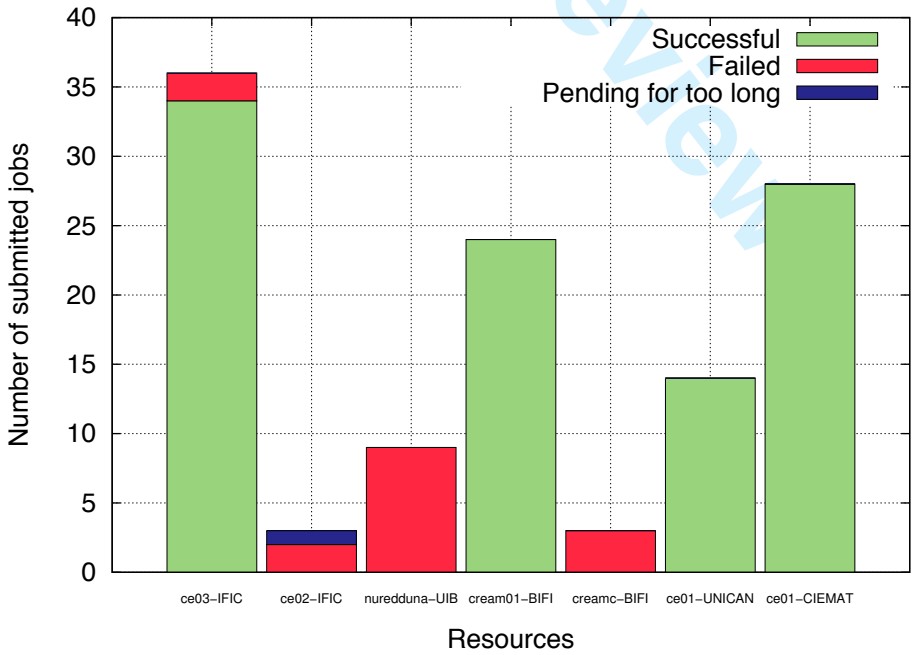

Figure 3. Successful and rescheduled jobs for each resource.

GridWay has been configured to reschedule failed jobs until they are successfully completed. Failure reasons include nodes temporarily down for maintenance or transient network problems [31]. Fig. 3 shows that 16 jobs failed, giving a failure rate of 13.68%. In fact, some resources (such as nuredduna at UIB or creamc at BIFI) presented configuration problems at the moment of performing the experiments, causing all the submitted jobs to fail and to be rescheduled to other sites. Moreover, GridWay implements an exponential linear back-off strategy, temporarily discarding for submission resources with failures. Submitted jobs that remain pending on the remote resource, without starting execution, longer than the threshold specified by the user (10 minutes) are also rescheduled to other resources. This is usually caused by a wrong number of free slots calculated by GridWay due to unpublished internal policies of LRMS, or due to outdated monitoring information.

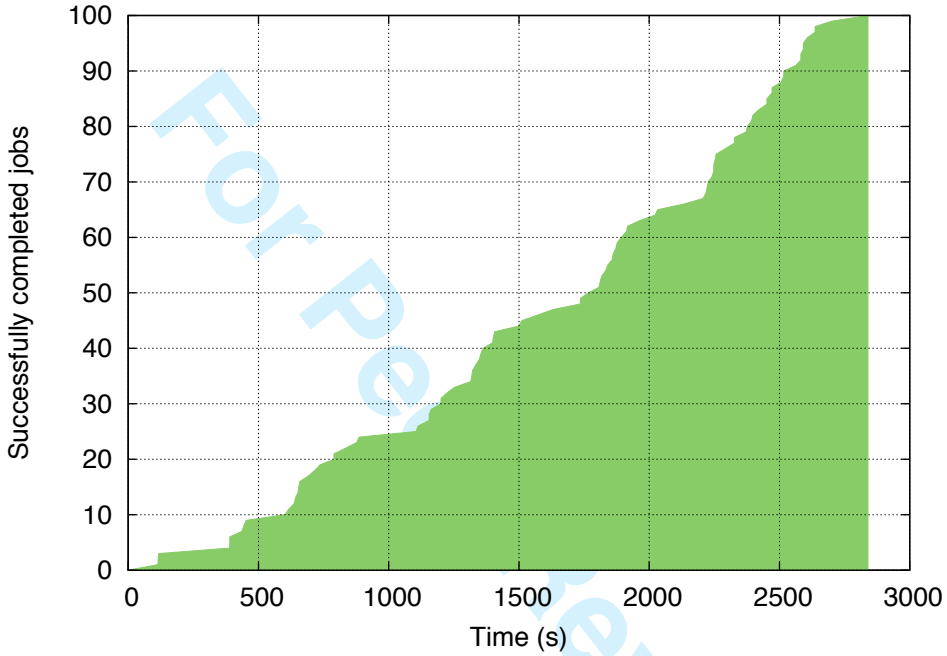

Figure 4. Jobs completed over time.

As shown in Fig. 4, the whole experiment took about 47 minutes, and all the jobs consumed around 9 hours and 27 minutes of aggregated CPU time. Execution time of the sequential application is approximately 3 hours and 36 minutes, when it runs on one of the fastest machines (ce03 at IFIC), thus resulting in a speedup of 4.6. However, this speedup is affected by overheads, such as service communications or unavoidable resource failures, that are always present in a grid [31]. The limitations introduced in order to avoid stressing the resources influence this result as well. Therefore, this speedup would of course be greater if more jobs were submitted, if jobs needed more time to complete, and if more resources had been used.

Table III shows the average time per job spent on overheads and execution. Overhead time represents the sum of the time waiting for the LRMS to run the job (queue waiting time), the time spent on transferring input/output files (transfer time) and other overheads produced by the middleware services or by GridWay. The execution time corresponds to the time spent while the job is being executed by the LRMS. Rescheduling times and other overheads due to failed jobs are not considered in the next table.

Jobs usually spent considerable time waiting for a resource to run. This means a low resource availability at the moment of performing the experiment, and causes a degradation of the performance. Also, since CREAM does not provide asynchronous notifications about the

Table III. Average time (s) per job spent for each computing resource.

HOSTNAME-ORG.	OVERHEAD	EXECUTION	TOTAL
ce03-IFIC	303.47	118.03	421.5
cream01-BIFI	460.92	100.46	561.38
ce01-UNICAN	182.93	365.79	548.71
ce01-CIEMAT	108.64	695.71	804.36

termination of job execution, the driver has to periodically query the execution service about the job status. This introduces a significant overhead, especially when jobs do not require much time to complete. Finally, average transfer time is approximately 30 seconds per job, which is very similar for all resources because sites are connected by research and education networks.

7.2. Using BES and GRAM resources in IGE

For the second experiment, a GridWay instance has been configured to access to the computing resources provided by IGE testbed. Next, the specific configuration of the MADs for this case is explained.

```
IM_MAD = static_gram:gw_im_mad_static:-l etc/ige.static:gridftp:gram5
IM_MAD = static_bes:gw_im_mad_static:-l etc/bes.static:dummy:bes
EM_MAD = gram5:gw_em_mad_gram5::rsl
EM_MAD = bes:GW_em_mad_bes::jsdl
TM_MAD = gridftp:gw_tm_mad_ftp:
TM_MAD = dummy:gw_tm_mad_dummy:-u gsiftp\://gridway.fdi.ucm.es
```

Two static information MADs (identified by `static_gram` and `static_bes`), based on files including host lists and descriptions, have been configured. The first information MAD provides information about the GRAM5 resources, and associates a GridFTP MAD and a GRAM5 MAD (identified by `gridftp` and `gram5`). The second information MAD provides information about a BES instance providing access to a single machine (bes at UCM), and associates a Dummy transfer MAD and a BES MAD (identified by `dummy` and `bes`).

The GRAM5 MAD gives access to the execution services provided by the IGE testbed. It uses the `rsl` generation function in order to describe the job requests according to the RSL specification. The BES MAD gives access to the BES instance. It uses `jsdl` as generation function to describe the job request according to the JSDL standard.

Finally, the GridFTP and Dummy MADs have been configured. On the one hand, the GridFTP driver is used to perform the data staging of jobs submitted to the GRAM resources at the IGE testbed. This MAD interfaces with the GridFTP servers on the remote hosts. On the other hand, the Dummy driver, using GridFTP in the client host, has been configured for managing the data staging of the BES endpoint.

Table IV shows the GRAM5 resources provided by the IGE testbed, while Table V shows the BES resource available for this experiment.

Table IV. GRAM5 resources provided by the IGE testbed.

HOSTNAME	ORGANIZATION	LOCATION	MHZ	Nodes	LRMS
gt5-ige	LRZ	Munich	2533	2	SGE
udo-gt01	TUDO	Dortmund	1995	44	PBS
ve	NIKHEF	Amsterdam	2993	8	Fork
gt1	EPCC	Edinburgh	1600	1	Fork
gt01	PSNC	Poznan	2328	1	PBS
gt-ige	UTCN	Cluj-Napoca	2000	4	Fork

Table V. BES resource.

HOSTNAME	ORGANIZATION	LOCATION	MHZ	Nodes	LRMS
bes	UCM	Madrid	1995	2	Fork

This experiment has been carried out under the same limitations than the previous experiment. As in the previous case, figures 5 and 6 show the results obtained. Fig. 5 shows the total number of jobs submitted to each host, presenting how many jobs were successfully completed or had to be rescheduled due to different causes. Fig. 6 shows the evolution of the completed jobs over time.

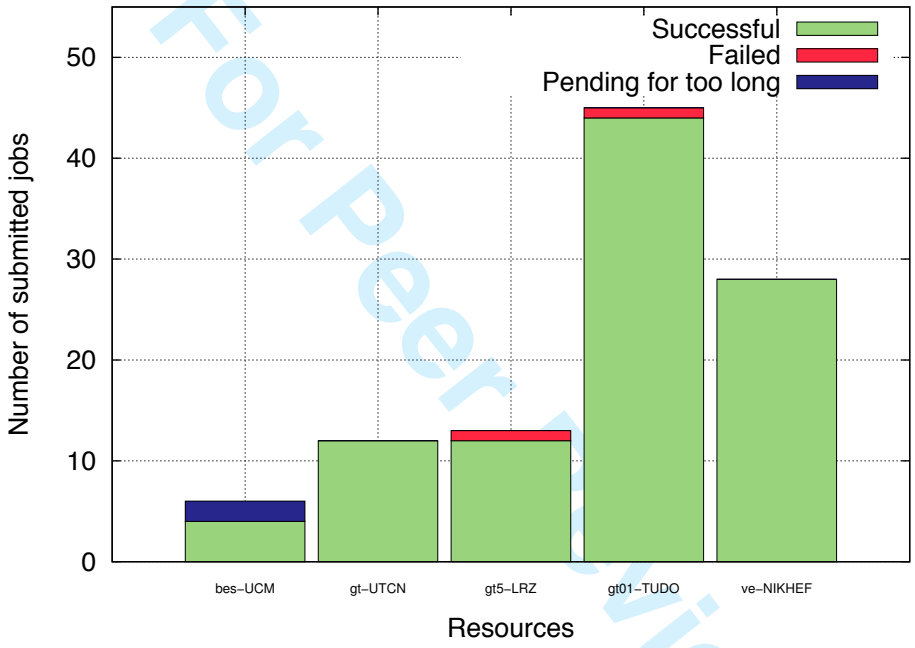

Figure 5. Successful and rescheduled jobs for each resource.

Fig. 5 presents a better failure rate (1.92%) than in the previous experiment, with only two failed jobs. This is mainly because the nodes provided by the IGE testbed are not very busy, reducing job failures due to performance degradations and so it favors less job reschedulings due to performance degradations. Most of the jobs were dispatched to GRAM resources, which have a higher rank than the BES endpoint.

The whole experiment took about 32 minutes, as Fig. 6 shows, and all the jobs consumed around 7 hours and 56 minutes of aggregated CPU time. Execution time of the sequential application is approximately 4 hours and 15 minutes, when it runs on one of the testbed fastest machines (gt5 at LRZ), thus resulting in a speedup of 7.97. This experiment required less time to complete since the IGE testbed is not overloaded. Then, jobs spend less time in the remote queue, and the computing nodes provide better performance conditions for running the jobs. The other reason to explain the better performance is that the GRAM driver is able of receiving notifications about the job state changes. This causes that the GridWay jobs are terminated as soon as the driver receives the notification that they are done. In the previous case, jobs managed by the CREAM driver have to wait to query the grid services about the job state returns a done state. This overhead is especially significant when jobs do not require much time to complete.

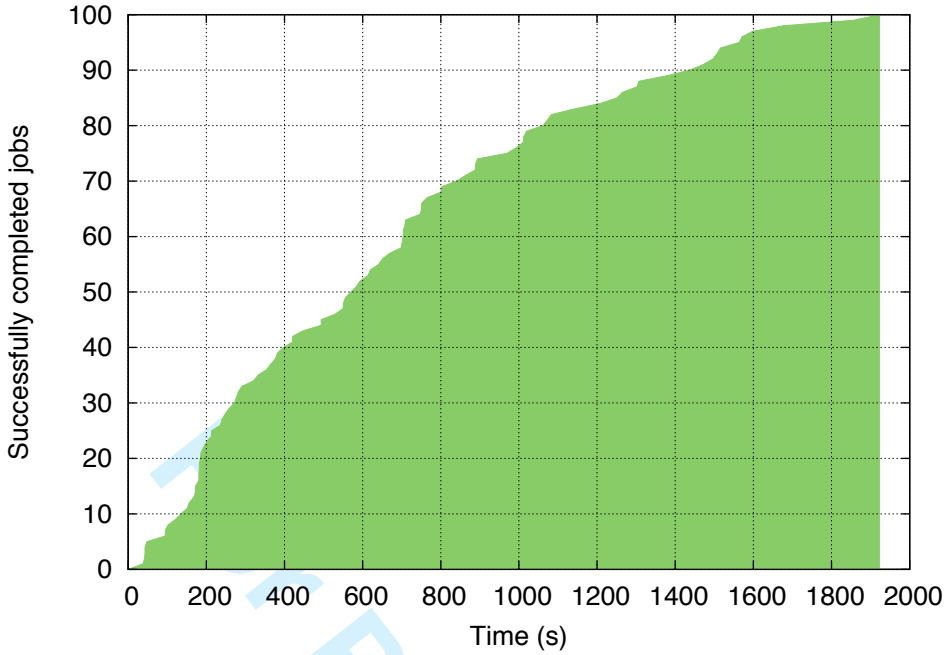

Figure 6. Jobs completed over time.

Table VI shows the average time due to overheads (queue and transfer times) and the average execution time. Rescheduling times and other overheads due to failed jobs are not considered on the next table.

Table VI. Average time (s) per job spent for each computing resource.

HOSTNAME-ORG.	OVERHEAD	EXECUTION	TOTAL
bes-UCM	97.25	268.75	366
gt-UTCN	41.34	494.08	535.42
gt5-LRZ	43.59	128	171.58
gt01-TUDO	41.75	223.57	265.32
ve-NIKHEF	44.78	364.75	409.54

The average computing time per job is considerably less than the previous case, which is basically due to jobs spent less time waiting for the LRMS to execute it, and that the GRAM driver is able of receiving notifications about the termination of job execution, as previously said. However, the BES specification does not support notifications, and so it introduces an important overhead for jobs submitted to the BES endpoint at UCM. Average transfer time is 34 seconds which is again quite similar for all resources as sites are also connected by research and academic networks.

8. CONCLUSIONS AND FORTHCOMING WORK

This paper has introduced the GridWay metascheduler and presented the new execution drivers that provide an unprecedented level of interoperation with grid infrastructures. These include a new driver that enables the submission and control of jobs through CREAM, which is the computing resource execution service of gLite, and another one that enables the submission and control of BES activities. The use of these drivers is completely transparent to the end-users. Moreover, the paper has shown how to configure GridWay to access to different grid infrastructures.

Also, a new BES interface for GridWay has been described. This enables the remote access to GridWay's metascheduling capabilities through a standard interface. The implementation of BES is provided by GridSAM, where a GridWay DRMAA Connector allows the interoperation of both systems.

Two experiments have been performed to evaluate the interoperation of GRAM2 and CREAM resources, using the EGI infrastructure, and the interoperation of the BES and GRAM5 resources using the IGE testbed. The results indicate a better performance in the second case, because the IGE testbed presented a lower usage load, and because GRAM provides notifications about job state changes, while CREAM and BES don't. Therefore, we are exploring ways to improve job state polling operations in CREAM and BES.

Other future work is to adapt the current DRMAA implementation provided by GridWay, based on the first version of the specification, according to the new DRMAA v2 specification. The new version is adapted to the latest improvements of LRMS and includes new features, such as resource monitoring, session persistence or a new sub-state concept similar to BES.

ACKNOWLEDGEMENTS

This research was supported by Consejería de Educación of Comunidad de Madrid, Fondo Europeo de Desarrollo Regional (FEDER) and Fondo Social Europeo (FSE) through MediaNet Research Program S2009/TIC-1468, by Ministerio de Ciencia e Innovación through research grant TIN2009-07146, and by European Union through IGE project RI-261560.

Finally, the authors would like to thank to the institutions participating on the IGE testbed and Ibergrid infrastructure for allowing the use of their computational facilities.

REFERENCES

1. Huedo E, Montero RS, Llorente IM. Grid architecture from a metascheduling perspective. *Computer* 2010; **43** (7): 51-56.
2. Foster I, Kesselman C. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications* 1997; **11** (2): 115-128.
3. Foster I. Globus Toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology* 2006; **21** (4): 513-520.
4. Laure E, Fisher SM, Frohner A, et al. Programming the grid with gLite. *Computational Methods in Science and Technology* 2006; **12** (1): 33-45.
5. Garzón JI, Huedo E, Montero RS, Llorente IM, Chacon P. Adaptation of a multi-resolution docking bioinformatics application to the grid. *Journal of Software* 2007; **2** (2): 1-10.
6. Ibarra A, Tapiador D, Huedo E, Montero RS, Gabriel C, Arviset C, Llorente, IM. On-the-fly XMM-Newton spacecraft data reduction on the grid. *Scientific Programming* 2006; **14** (2): 141-150.
7. Rodriguez-Pascual M, Guasp J, Castejon F, Rubio-Montero AJ, Llorente IM, Mayo R. Improvements on the fusion code FAFNER2. *IEEE Transactions on Plasma Science* 2010; **38** (9): 2102-2110.
8. Badia RM, Du D, Huedo E, Kokossis A, Llorente IM, Montero RS, de Palol M, Sirvent R, Vazquez C. Integration of GRID superscalar and GridWay metascheduler with the DRMAA OGF standard. *Proceedings of the 14th International Conference on Parallel Processing (Euro-Par 2008)* (*Lecture Notes in Computer Science*, vol. 5168), Luque E, Margalef T, Benítez D (eds.). Springer: Berlin, 2008; 445-455.
9. Riedel M, Laure E, Soddemann T, et al. Interoperation of world-wide production e-Science infrastructures. *Concurrency and Computation: Practice and Experience* 2009; **21** (8): 961-990.
10. Rajic H, Brobst R, Chan W, et al. Distributed Resource Management Application API Specification. Technical Report GFD-R.022, Open Grid Forum, 2004.
11. Tröger P, Rajic H, Haas A, Domagalski P. Standardization of an API for distributed resource management systems. *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*, May 2007. IEEE Computer Society: Washington, DC, 2007; 619-626.
12. Anjomshoaa A, Brisard F, Drescher M, Fellows D, Ly A, McGough S, Pulsipher D, Savva A. Job Submission Description Language (JSDL) Specification. Technical Report GFD-R.056, Open Grid Forum, 2005.
13. Foster I, Czajkowski K, Ferguson DF, Frey J, Graham S, Maguire T, Snelling D, Tuecke S. Modeling and managing state in distributed systems: The role of OGSI and WSRF. *Proceedings of the IEEE* 2005; **93** (3): 604-612.
14. Huedo E, Montero RS, Llorente IM. A modular meta-scheduling architecture for interfacing with pre-WS and WS grid resource management services. *Future Generation Computer Systems* 2007; **23** (2): 252-261.
15. Huedo E, Montero RS, Llorente IM. A recursive architecture for hierarchical grid resource management. *Future Generation Computer Systems* 2009; **25** (4): 401-405.
16. Cofiño AS, Blanco C, Fernández-Quiruelas V. Aggregation of grid and HPC resources for running huge experiments in climate and weather prediction. *8th European Geosciences Union General Assembly* (*Geophysical Research Abstracts* , vol. 13), April 2011.

17. Foster I, Grimshaw A, Lane P, et al. OGSA Basic Execution Service. Technical Report GFD-R.108, Open Grid Forum, 2008.
18. Snelling D, van den Berghe S, von Laszewski G, Wieder P, Breuer D, MacLaren J, Nicole D, Hoppe HC. A UNICORE-Globus interoperability layer. *Computing and Informatics* 2002; **21** (4): 399-411.
19. Kacsuk P. P-GRADE portal family for grid infrastructures. *Concurrency and Computation: Practice and Experience* 2011; **23** (3): 235-245.
20. Kertész A, Kacsuk P. Grid meta-broker architecture: Towards an interoperable grid resource brokering service. *Proceedings of the 12th International Conference on Parallel Processing (Euro-Par 2006)* (*Lecture Notes in Computer Science*, vol. 4375), Lehner W, Meyer N, Streit A, Stewart C. (eds.). Springer: Berlin, 2007; 112-115.
21. Bobroff N, Fong L, Kalayci S, Liu Y, Martínez JC, Rodero I, Sadjadi SM, Villegas D. Enabling interoperability among meta-schedulers. *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2008)*, May 2008. IEEE Computer Society: Washington, DC, 2008; 306-315.
22. Assunção MD, Buyya R, Venugopal S. Intergrid: A case for internetworking islands of grids. *Concurrency and Computation: Practice and Experience* 2008; **20** (8): 997-1024.
23. Agarwal A, Ahmed M, Berman A, et al. GridX1: A Canadian computational grid.*Future Generation Computing Systems* 2007; **23** (5): 680-687.
24. Crouch S. Interoperability with GridWay. Grid Interoperation Now (GIN) Community, *34th Open Grid Forum*, March 2012.
25. Herrera J, Huedo E, Montero RS, Llorente IM. Developing grid-aware applications with DRMAA on Globus-based grids. *Proceedings of the 10th International Conference on Parallel Processing (Euro-Par 2004)* (*Lecture Notes in Computer Science*, vol. 3149), Danelutto M, Vanneschi M, Laforenza D. (eds.). Springer: Berlin, 2004; 429-435.
26. Leal K, Huedo E, Llorente IM. Performance-based scheduling strategies for HTC applications in complex federated grids. *Concurrency and Computation: Practice and Experience* 2010; **22** (11): 1416-1432.
27. Aiftimiei C, Andreetto P, Bertocco S, et al. Design and implementation of the gLite CREAM job management service. *Future Generation Computer Systems* 2010; **26** (4): 654-667.
28. Aiftimiei C, Andreetto P, Bertocco S, et al. Using CREAM and CEMonitor for job submission and management in the gLite middleware. *Journal of Physics: Conference Series* 2010; **219** (6): 062001.
29. Smith C, Kielmann T, Newhouse S, Humphrey M. The HPC basic profile and SAGA: standardizing compute grid access in the Open Grid Forum. *Concurrency and Computation: Practice and Experience* 2009; **21** (8): 1053-1068.
30. McGough AS, Leeb W, Dasc S. A standards based approach to enabling legacy applications on the grid. *Future Generation Computer Systems* 2008; **24** (7): 731-743.
31. Huedo E, Montero RS, Llorente IM. Evaluating the reliability of computational grids from the end user's point of view. *Journal of Systems Architecture* 2006; **52** (12): 727-736.