Federation of TeraGrid, EGEE and OSG Infrastructures through a Metascheduler[☆]

Constantino Vázquez, Eduardo Huedo, Rubén S. Montero, Ignacio M. Llorente

Departamento de Arquitectura de Computadores y Automática
Facultad de Informática
Universidad Complutense de Madrid, Spain
December 11, 2008

Abstract

Since its conception, Grid technology concerned itself with interoperability between heterogeneous computers by providing a middleware layer to abstract underlying resource characteristics. Unfortunately there is no agreement today on a common set of standards and, therefore, different conceptions lead to different middleware implementations. That effectively rendered interoperability between grid infrastructures to be a complex issue, which was what originally sprang the idea of Grid. In this paper we present technologies to achieve interoperation (i.e. interoperability not based on standards) between sites and show its feasibility, aiming to provide a mid-term solution to the federation problem while the promise of interoperability through standards becomes a reality. Using interoperation techniques and bringing them together in a common federation component, the GridWay metascheduler, we are able to offer common access to well known grid infrastructures. This approach is demonstrated in the performance evaluation of the execution of one benchmark of the NAS Grid Benchmark suite.

[☆]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 MEDIANET Research Program S2009/TIC-1468, by Ministerio de Ciencia e Innovación, through the research grant TIN2009-07146, and by the European Union through the research grant EGEE-III Contract Number 22667.

Email addresses: tinova@fdi.ucm.es (Constantino Vázquez), ehuedo@fdi.ucm.es (Eduardo Huedo), rubensm@dacya.ucm.es (Rubén S. Montero), llorente@dacya.ucm.es (Ignacio M. Llorente)

Key words: grid, interoperability, federation, scalability, metascheduler

1. Introduction

A growing number of grid infrastructures and middlewares are being developed and deployed, favored by the high expectations raised by Grid [1, 2] computing. Some of the most successful infrastructures (e.g. TeraGrid[1], EGEE[2] or Open Science Grid[3]) are used at production level, enabling resource sharing and collaboration between scientific institutions geographically apart. Nevertheless, if we want to move forward and aggregate resources from these infrastructures, we face several problems.

It is not really surprising finding here the same problems found in the early stages of grid technology. The goal is still the same: harness resources independently of their software stack, geographical disposition or administration domains; albeit scaled up the ladder a bit: we are trying to interoperate grids. These grids have potentially different middleware stacks which make them incompatible, and even those who use the same middleware have slightly different deployment characteristics that makes them non-interoperable.

There are many examples in the literature on how to achieve federation between grids with different interfaces in order to achieve the so-longed-for world wide grid. Interoperability through the use of standard and open interfaces is one of the defining characteristics of grid technologies [3]. There are ongoing projects that address the definition of these standard interfaces, like for example SAGA [4], an initiative to provide grid applications with a common standard API so they can be built in a middleware-agnostic way. This is an example of a solution for interoperability from the application point of view. There are standardization efforts like the Job Submission Description Language (JSDL), designed to describe jobs to be run on the grid. Others, like the Basic Execution Service (BES), focuses on how to send requests to initiate, monitor, and manage computational activities. These two standards (JSDL and BES) are used in the HPC Basic Profile [4]. Both SAGA and HPC Profile efforts stem from the Open Grid Forum (OGF[4]), an organization that promotes the creation of standards for interoperability.

[1] http://www.teragrid.org/
[2] http://www.eu-egee.org/
[3] http://www.opensciencegrid.org/
[4] http://www.ogf.org/

But while other needed standards are being defined or current ones consolidated, there is another set of efforts that focuses on what is available nowadays and tries to provide federation solutions for the short term. This is exactly the aim of the Grid Interoperation Now [5] (GIN) group of the OGF, some of the projects mentioned from now onwards participate in it. Following this interoperation philosophy, we can remark efforts to interoperate different grid middlewares (for example, Globus and UNICORE [6]) and approaches based on portals and workflows [7]. In this line, it is also interesting to see efforts like InterGrid [8], which proposes the creation of InterGrid Gateways (IGGs) to interconnect the existing different grid islands, or GridX1 [9], which can be accessed as another resource of LCG. We can find other works based on the same idea of interoperation using current interfaces, for instance by introducing the concept of a meta-broker component [10], which will aid to access resources outside one grid domain. Moreover, and complementary to the direction taken by this paper, there are various works enabling interoperability between existing metaschedulers [11]. There is even an OGF group devoted to this research line, the Grid Scheduling Architecture (GSA) research group.

In this paper, we will show how to uniformly access grid resources from different infrastructures and how the tool used for this solution (the GridWay metascheduler [12]) is production ready, i.e., it scales. We will use *federation* to describe this aggregation of resources, and show how it is defined by two intrinsic characteristics: *interoperation* and *scalability*, although there are other, less technical issues out of the scope of this paper. Our solution offers unprecedented levels of flexibility for constructing any kind of grid infrastructure.

The remainder of this paper conforms with the following structure. Section 2 presents a strategy to achieve federation, while Section 3 describes the tool involved in the interoperation process and Section 4 shows how it scales and therefore is suitable for federation. Afterwards, Section 5 explains the output of the different federation experiments (coming from different federation approaches) and comments on their differences. Finally, Section 6 outlines the main conclusions that can be drawn from this work and sets plans for future work on this matter.

2. Interoperation for Federation

Reaching agreement between infrastructures on which interface to use to achieve interoperability takes time [13]. Driven by this factor, the OGF makes a clean distinction between *interoperability*, or the native ability of grids and grid technologies to interact directly via common open standards, and *interoperation*, or the set of techniques to get production grid infrastructures to work together in the short term.

Hence, we can think of *interoperation* as a more immediate solution for the collaboration between two or more heterogeneous grids. On the other hand, *interoperability* focuses on the big picture and tries to bring together technologies that implement the grid infrastructure by means of standardization (like, for example, SAGA or BES do). It is clear that this is not a feat that can be achieved without a significant amount of effort and, more important to the point being made here, time. Thus, the need to provide interoperation and the justification of the GIN group within the OGF.

Since most common open standards to provide grid interoperability are still being defined and only a few have been consolidated, grid interoperation techniques, like adapters and gateways, are needed. An adapter is, according to different dictionaries of computer terms, a device that allows one system to connect to and work with another. On the other hand, a gateway is conceptually similar to an adapter, but it is implemented as an independent service, acting as a bridge between two systems. The main drawback of adapters is that grid middleware or tools must be modified to insert the adapters. In contrast, gateways can be accessed without changes on grid middleware or tools, but they can become a single point of failure or a scalability bottleneck [13].

Taking this techniques into account, there are three different ways in which interoperation can be achieved:

- *Common interfaces.* This is the most straightforward way of interoperation. It requires that all sites involved in the federation share the same interfaces, so interoperation between them becomes just an administrative task regarding authorization and other policies. A single point of entry (i.e. a portal) can be easily built, since it can be used to access all the sites using the same mechanism, although, as will be seen in Section 5, sharing the same middleware stack or the same interfaces does not guarantee the ability of having exactly the same way to access them, due to, for example, different configurations that the middleware is subject to.

- *Adapters.* Although sharing the same interfaces is the ideal way to achieve interoperation, sometimes there are different middlewares deployed in the sites to be federated, and it is unfeasible to unify them for a variety of reasons (politics, time constraints or ongoing migration or upgrades). This can be seen as the consequence of not having an accepted standard, and therefore, lack of interoperability. One possible solution to federate these different sites is to build a portal that uses different components (to submit jobs, gather information, transfer files, etc) to interface these sites. These components are designed specifically for a particular middleware stack or even version, and we can call them adapters.

- *Gateways.* Another way to achieve interoperation consists in encapsulating one or more sites under one single resource accessed through one interface. The interoperation problem can then be reduced to the *Common interfaces* scenario. This encapsulation acts translating the requests from the portal to requests that the underlying sites can understand, and conveying the results to the originator of the job request. This technique is conceptually similar to network *gateways* and is especially suited for situations where it is not possible to modify higher level services.

Therefore we must wonder whether interoperation is a necessary and *sufficient* condition for grid federation. We argue that this is not the case, since there are even non-technical issues that we are not taking into account, like for example political matters (when and how a user should use or not a given infrastructure) and operational issues (which software is installed in the execution nodes, for example). Nevertheless, in Section 5 it is shown how two grid infrastructures with common interfaces (as TeraGrid and Open Science Grid having the same execution service interface, Globus GRAM) are not necessarily federated, but interoperation is needed because their execution model is different (for example, their storage model is very dissimilar). Thus interoperation is a necessary condition for federation.

Considering interoperation enough for federation leaves the key factor out of the equation. This factor is indeed scalability, it is important to know if the component that performs the federation actually scales. This question is tackled in Section 4, where we are going to measure the scalability of the component we are using for federation: the metascheduler. Reached this

point we claim that true federation is achieved, among other less technical issues, by the conjunction of two key factors: *interoperation* and *scalability*

3. The GridWay Metascheduler

The GridWay metascheduler enables large-scale, reliable and efficient sharing of computing resources (clusters, computing farms, servers, supercomputers...), managed by different LRM (Local Resource Management) systems, within a single organization (enterprise grid) or scattered across several administrative domains (partner or global grid).

GridWay is composed of several modules as seen in Figure 1. First we have the GridWay daemon, which is the core that coordinates the whole job life cycle process by means of a state machine. It uses a set of Middleware Access Drivers (MADs), basically independent processes that talk with the daemon using the standard I/O streams. They are used to perform execution (for job submission), transfer (for data staging) and information (for resource discovery and monitoring) tasks; several MADs of each type can be loaded and used simultaneously. The last component in which we can decompose GridWay in is the scheduler, which also lives in a separate process, communicating with the daemon also through the standard streams, and is in charge of the job allocation in the resources.

GridWay exposes two ways of interfacing with the core. One is a powerful command line interface (CLI), featuring commands to manage jobs and computing resources. But GridWay can be used programatically with the Distributed Resource Management Application API (DRMAA), which is an OGF standard. In this way, applications can be built abstracted from the heterogeneous resources where it can be executed.

4. Experiments: Metascheduler scalability

In this section we will show experiments designed to demonstrate GridWay's scalability and, hence, its ability to perform the role of federation component. To measure the scalability of GridWay, we set up an experiment that tries to saturate all GridWay components by submitting 10,000 jobs.

The resources used for the testbed are located in the Universidad Complutense de Madrid (UCM) local grid, so issues like network saturation can be more controlled. These resources are Intel Pentium 4 3.2GHz with 2GB

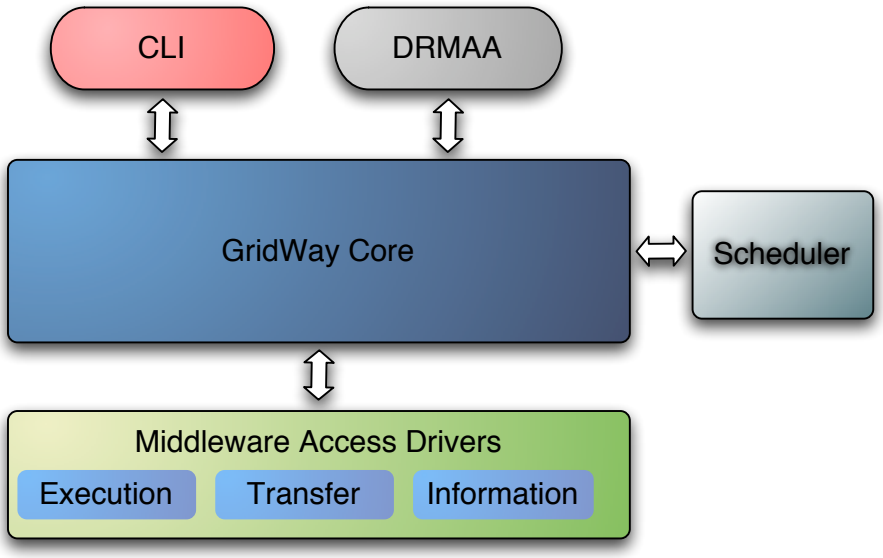

Figure 1: GridWay high level architecture.

of RAM that communicate with each other over a 100 LAN network, running Debian Etch. The testbed for the scalability test uses four computing elements with the aforementioned specifications, served by GRAM pre-Web Services (preWS). The GridWay client is also a machine with the same characteristics.

GridWay was configured using a static information provider. The latter allowed to configure the four hosts for GridWay as if having two hundred free slots each managed by a Fork job manager. The performance benefit of using the static information MAD as opposed to a dynamic one is negligible due to the small number of hosts. GridWay scalability for discovering and monitoring is quite good, particularly with the new multithreaded MAD for the Monitoring and Discovery Service of the Globus Toolkit 4.0 (MDS4). Moreover, GridWay implements throttling techniques to avoid saturation that could occur if all the known hosts were to be monitored simultaneously.

The GridWay instance was configured so it submits jobs at the following rate:

- In each iteration, the scheduler submits a maximum of 60 jobs (DISPATCH_CHUNK = 60).
- Each iteration of the scheduler happens with a 3 minutes interval

(SCHEDULING_INTERVAL = 180).

- There is no limit on how many jobs per user can be running at the same time (MAX_RUNNING_USER = 0).
- There is no limit on the maximum number of jobs that the scheduler submits to a given resource (MAX_RUNNING_RESOURCE = 0), so ideally all the two hundred slots would be used per host.

Jobs being sent in this experiment consist basically of a task sleeping from 1800 to 5400 minutes. Each job gets their arguments according to a piecewise linear function, defined in Equation 1

$$f(i) = 1800 + (20i)\%(5400 - 1800), \tag{1}$$

where i is the job index given by GridWay, ranging from 0 to 9,999.

This task doesn't make use of the CPU, since we are not interested in measuring a computer saturation but rather GridWay's. The file staging carried by each job involves the creation of a directory in the prolog state and the removal of that directory in the epilog state, and also two set of files, which amount to 12 KB for the transfer of input files and 16 KB for the transfer of output files.

The first interesting result of the test is the time that GridWay needs to accept, sequentially, 10,000 jobs. The average time in seconds for this is 226 seconds, i.e., a bit less than four minutes. This contrasts strongly with the 2 hours needed by the gLite Workload Management System (WMS) to accept 500 jobs [14].

We can see in Figure 2 the disk consumption of GridWay. For ten thousand *completed* jobs the disk consumption gets to 660,084 KB, and that gives an average of approximately 66 KB per job. It is worth noting that this number includes all the logging that GridWay is capable of producing, including information for debugging, being the actual production value significantly less.

Figure 3 is more revealing as it shows the percentage of memory consumption for all GridWay components and the total add between all of them. We can see how the Transfer MAD gets most of the memory, but it is important to note how the GridWay daemon and the scheduler are less demanding in memory consumption. Two sudden increases in Transfer MAD's memory consumption can be seen around minutes 300 and 600, probably due to memory leaks triggered by the handling of job errors.

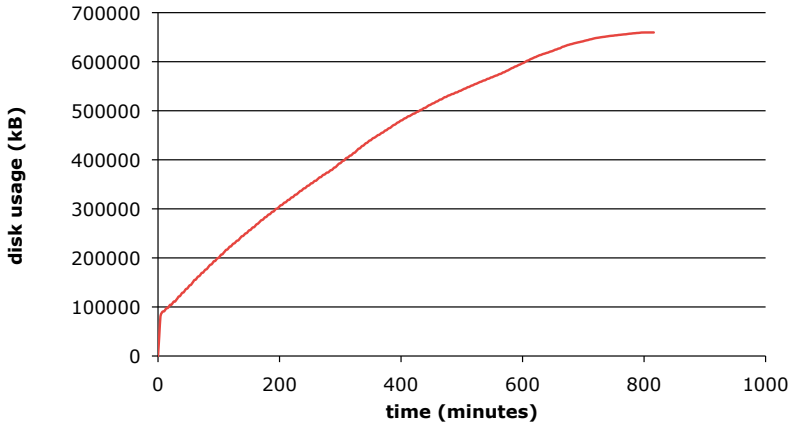

Figure 2: Disk consumption by all GridWay components, including logs and accounting information.

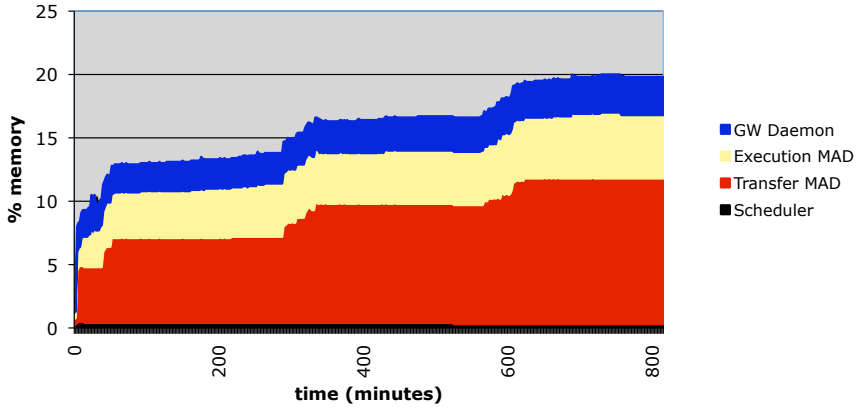

Figure 3: Memory consumption by GridWay split across its different components.

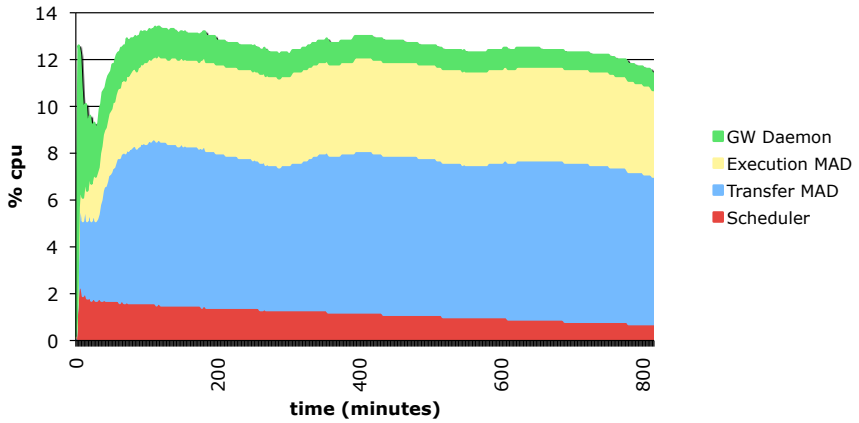

Figure 4: CPU consumption by GridWay's components.

Figure 4 shows the CPU consumption. We can see here a similar trend, where the MADs take the most CPU time and the GridWay daemon and the scheduler don't take that much. It is worth noting how the scheduler CPU consumption reduces when there are fewer pending jobs to be scheduled. It is therefore clear from these two last figures that GridWay's CPU and memory consumption are particularly low, if we take into account that it is managing 10,000 jobs.

Figure 5 depicts how the jobs get themselves from the pending to the done state, maintaining an acceptable average rate of 800 jobs running. It is interesting to see how there are two backward movements in the number of completed tasks around minutes 300 and 600, matching the CPU and memory consumption increase of the transfer MAD. This is coherent with the interpretation of error conditions around those times (increase of pending jobs indicates that running jobs did fail), and shows the robustness of GridWay in how it managed to complete all the jobs successfully.

5. Experiments: Interoperation

In order to demonstrate the feasibility of both the adapters and the gateway solutions for interoperation, two different scenarios were deployed and jobs were sent using GridWay to a set of resources gathered from three major

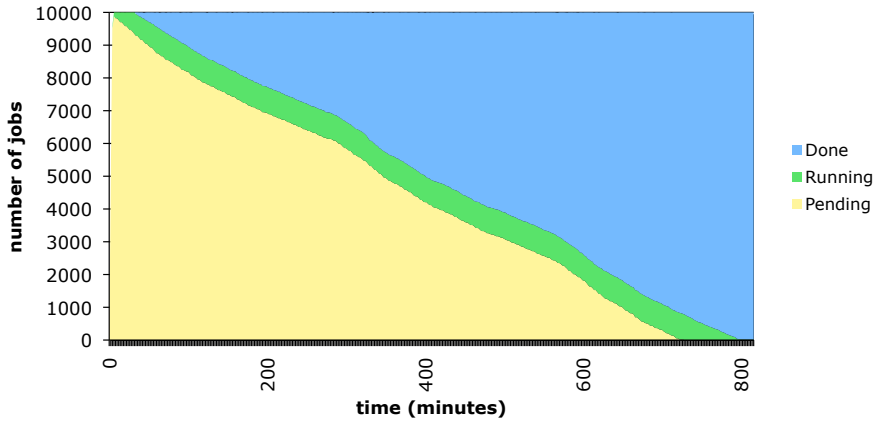

Figure 5: Job states distribution against time.

grid infrastructures: Open Science Grid, TeraGrid and EGEE. Additionally, resources from a local UCM cluster were used as well. Three iterations of two hundred jobs each were submitted in each of the two scenarios in order to show their feasibility. For both these experiments we used the Embarrassingly Distributed (ED) benchmark from the NAS Grid Benchmark suite [15].

Figure 6 depicts the architecture of both experiment interfaces. The adapters scenario presents a flat line-up of the available resources, all of them accessed by different adapters (dotted arrows represent pre Web Services (preWS) interfaces, solid arrows represent Web Services (WS) interfaces). On the other hand, the gateway scenario distributes resources in two layers, separating resources by their accessing adapters: all resources in the upper level are accessed using common interfaces, as are those on the lower level, albeit this time the interfaces are different. The GridWay configuration needed to set up the scenarios and the results obtained are presented in the following two subsections.

5.1. Adapters Scenario

To address the adapters scenario, the GridWay metascheduler was configured to access four different infrastructures. Not only different interfaces were the problem, but also different versions of the same middleware posed their own issues. Furthermore, different configurations of even the same version of

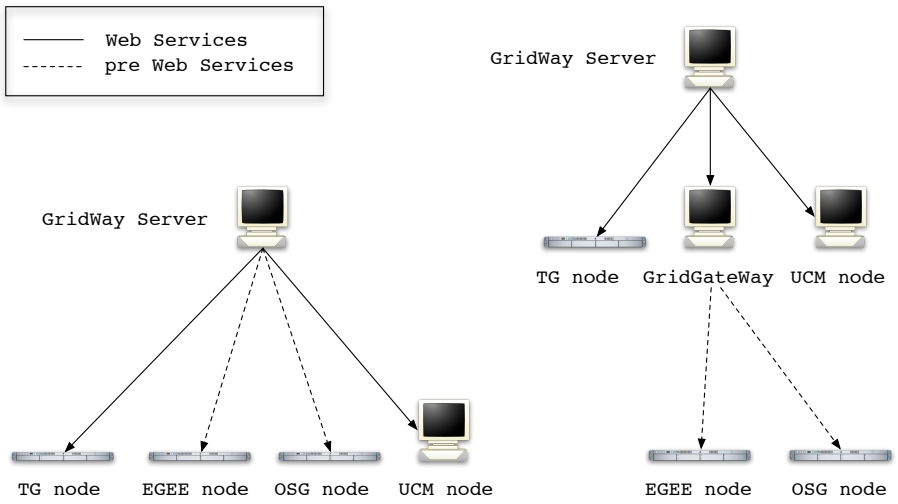

Figure 6: Adapters scenario vs Gateway scenario

the same middleware stacks are troublesome for the correct interoperation of the four infrastructures.

GridWay's adapters are the Middleware Access Drivers (MADs), enabling the metascheduler to access different infrastructures simultaneously and seamlessly. GridWay has evolved over time to take advantage of the adapters technique, it is interesting to see its evolution, from a first tentative effort to harness both EGEE and the IRISGrid infrastructures [16], to the seminal work that produced its full blown current modular architecture [17], motivated by the interoperation between grid resource management services provided by Globus. There are three types of adapters: execution, transfer and information MADs. Details about them and their configuration follows:

- *Open Science Grid*: This infrastructure offers two versions of the Globus Toolkit, deployed using VDT[5]. The particularity of the OSG configuration-wise with respect to a standard Globus install is that it uses the non standard 9443 port to offer the WS created by the Globus container, so GridWay MADs have to be configured accordingly[6].

- *TeraGrid*: Again, two versions of Globus are offered. Configuration

[5]http://vdt.cs.wisc.edu/

[6]http://www.gridway.org/documentation/howtos/osghowto.pdf

for the file transfer was different than in other infrastructures since the Storage Element (SE) was a separate machine (sharing homes with the Computing Element), thus the SE_HOSTNAME attribute of GridWay was used to direct staging to the SE node[7].

- *EGEE*: This infrastructure uses the gLite middleware stack, which is based on Globus preWS. The preWS MADs are used, but there are a few particularities of the EGEE that requires a different way of file staging. Mainly, worker nodes of EGEE don't share their home folders with the front-end. Instead, each has an outbound connection. Thus, the weight of the file staging is left to the wrapper script, which is in charge of staging the files and executing the job[8]. An information MAD for the Berkely Database Information Index (BDII) (which is basically a LDAP server running on port 2170) using the GLUE scheme is used for host monitoring.

- *UCM:* This is dsa-research.org group local cluster at Universidad Complutense de Madrid. Probably due to the fact that the Globus WS MADs were developed against this very cluster, no extra configuration was needed for GridWay to access it.

A testbed was prepared in order to perform the experiments in the adapters scenario. A demonstration in TeraGrid07 featured GridWay accessing several resources from the three infrastructures and resources from UCM, as shown in Figure 7. For illustrative purposes, one resource from each infrastructure was chosen for the experiments reported in this paper, so they could be performed in a more controlled environment. A complete resource listing for the adapters scenario can be seen in Figure 8. To show different adapters in action, WS MADs were used to access the TeraGrid (with Host Identifier , HID, 0 in the figure) and the UCM (HID 2) infrastructure, while the preWS MADs were chosen to access the Open Science Grid (HID 1) and the EGEE (HID 3) infrastructure. GridWay is able to use MADs from toolkits of both version 4.0.x and 4.2.x of the Globus Toolkit, solving neatly the incompatibility between them and allowing a smooth upgrading process.

Figure 9 shows the number of jobs against the infrastructures where the jobs were executed. There is an approximately even distribution between our

[7]http://www.gridway.org/documentation/howtos/tghowto.pdf
[8]http://www.gridway.org/documentation/howtos/egeehowto.pdf

```
HID PRI OS              ARCH   MHZ %CPU  MEM(F/T)    DISK(F/T)  N(U/F/T) LRMS              HOSTNAME
0   20  Linux2.4.21-32. i686  2665  189  964/2006  62787/73964     0/2/1 jobmanager-fork   atlas.dpcc.uta.edu
1   20  Linux2.4.21-37. athlo 1991  110 1214/2009  62680/71679     0/1/1 jobmanager-condor ce01.cmsaf.mit.edu
2   20  Linux2.4.21-20. i686  2665  388 1246/2006  70846/99795     0/1/1 jobmanager-condor fiupg.ampath.net
3   20  Linux2.4.27-1-3       1595   95   381/884    6350/7525     0/1/1 jobmanager-condor grid.physics.purdue.edu
4   20  Linux2.6.9-42.0 x86_6 2592  200 3041/3901  12040/13770     0/1/1 jobmanager-condor osg.hpcc.nd.edu
5   75                                                0/89/300 PBS              tg-grid.uc.teragrid.org
6   1   ScientificSLSL  i686  1200    0 1024/1024          0/0   0/51/75 jobmanager-pbs    lcg2ce.ific.uv.es
7   1   ScientificSL4   i686   866    0   513/513          0/0   0/22/22 jobmanager-lcgpbs ramses.dsic.upv.es
8   1   ScientificSL4   i686  2800    0 1024/1024          0/0 0/152/158 jobmanager-lcgpbs lcg-ce.usc.cesga.es
9   1   ScientificSLBer i686  3000    0 1024/1024          0/0  0/49/108 jobmanager-lcgpbs ce2.egee.cesga.es
10  1   ScientificSLBer i686  4000    0 1024/1024          0/0    0/0/24 jobmanager-lcgpbs ifaece01.pic.es
11  25                                                0/214/219 Condor          nest.phys.uwm.edu
12  25                                                   0/6/11 Condor          osg-itb.ligo.caltech.edu
13  90  Linux2.6.17-2-6  x86  3216    0   45/2027 70824/118812     0/0/2 Fork             cygnus.dacya.ucm.es
14  90  Linux2.6.17-2-6  x86  3216  181  681/2027  98861/11881     0/2/2 Fork             draco.dacya.ucm.es
15  90  Linux2.6.18-4-a x86_6 2211  100  954/1003  77081/77844     0/3/4 PBS              hydrus.dacya.ucm.es
16  90  Linux2.6.18-4-a x86_6 2211  100  776/1003  76428/77844     0/5/5 SGE              aquila.dacya.ucm.es
```

Figure 7: Resources accessed by GridWay in the Teragrid07 demo.

```
HID PRI  OS              ARCH   MHZ %CPU MEM(F/T)   DISK(F/T)  N(U/F/T)  LRMS              HOSTNAME
0   1                                                   0/5/5  PBS              tg-grid.uc.teragrid.org
1   1    Linux2.4.21-32. i686  2665  189 964/2006 62787/73964   0/10/10  jobmanager-condor cmsgrid01.hep.wisc.edu
2   1    Linux2.6.24-17  x86_6 1995  100   18/499 10169/21817     0/5/5  SGE              aquila.dacya.ucm.es
3   1    ScientificSL4   i686   866    0  513/513         0/0   0/21/22  jobmanager-lcgpbs ramses.dsic.upv.es
```

Figure 8: Resources for the Adapters scenario, as provided by the `gwhost` command.

local cluster (*UCM*), the Open Science Grid preWS resource (*OSG preWS*) and the TeraGrid WS one (*TG WS*), while the *EGEE* resource shows a smaller ratio of jobs completed. This is due to the chosen EGEE site for the EGEE, ramses.dsic.upv.es, having lower computing power (both CPU and memory) than the resource chosen for this experiment in the other infrastructures.

Average suspension times can be seen in Table 1, where suspension time is the time a job spends waiting on the remote queue to be executed, and they show in this case the dynamic nature of the grid. Since resources are not completely devoted to our experiment, we thus experience disparity of the measured times between iterations. Here we can see the *OSG preWS* resource rendering different suspension times and how they directly affect to its amount of taken jobs. From the comparatively lower times shown by the EGEE resource we can deduce that it is probably less occupied than the *OSG preWS* resource, but its comparatively slowness prevents it to increase the ratio of completed jobs.

Productivity in Table 2 measures the amount of jobs taken by each infrastructure per hour. As expected, the *EGEE* resource shows a much lower productivity, again due to the lower performance of the chosen resource,

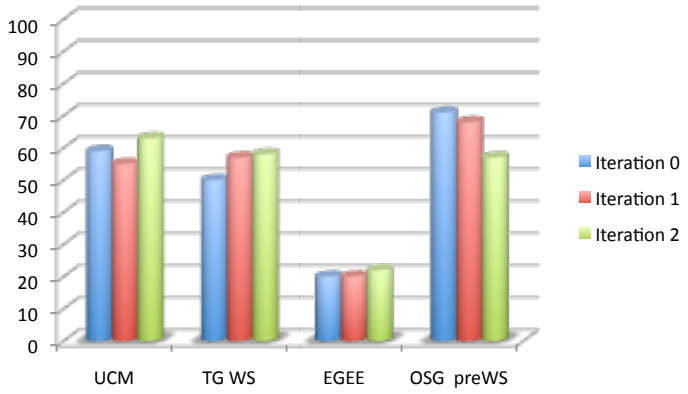

Figure 9: Job distribution across infrastructures: Adapters Scenario

Iteration #	UCM	TG WS	EGEE	OSG preWS
1	208.59	35.84	86.05	90.47
2	218.43	16.4	85.65	107.16
3	225.52	16.39	84.14	194.85

Table 1: Average suspension times as experienced by jobs from Adapters Scenario

Iteration #	Total	UCM	TG WS	EGEE	OSG preWS
1	125.78	37.10	31.44	12.57	44.65
2	144.92	39.85	41.30	14.49	49.27
3	122.7	38.65	35.58	13.49	34.96

Table 2: Productivity achieved (total and per resource) in the Adapters Scenario

```
HID PRI OS              ARCH   MHZ %CPU  MEM(F/T)   DISK(F/T) N(U/F/T) LRMS  HOSTNAME
0   1                                                  0/5/5 PBS   tg-grid.uc.teragrid.org
1   1   Linux2.6.24-17  x86_6 1995  100    18/499 10169/21817   0/5/5 SGE   aquila.dacya.ucm.es
2   1                                                0/26/32 GW    cepheus.dacya.ucm.es
```

Figure 10: Resources for Gateway Scenario upper level, as seen by `gwhost` in the GridWay server

while the other three infrastructures show more even productivity. It is interesting to see how the suspension time affects the productivity in the case of the *OSG preWS* resource, and in turn affects the amount of jobs taken shown in Figure 9. This can be explained if we take into account that an increase in the suspension time of a particular resource causes its designated jobs to wait more time in the queue, hence the productivity decreases (less jobs completed by the hour) and thus the amount of completed jobs also decreases.

5.2. Gateway Scenario

The gateway scenario is configured in a hierarchical manner. In the upper level, GridWay accesses resources for the TeraGrid and the UCM local cluster. Configuration details for these two infrastructures are identical as in the adapters scenario. A special case is *cepheus*, a resource encapsulating another instance of GridWay through a preWS or WS GRAM interface. This is what we call a GridGateWay [18]. Resources for this upper level can be seen in Figure 10. Information corresponding to *cepheus* only accounts for the aggregation of free and used nodes of all the resources being encapsulated in the lower level.

At a lower level we have an encapsulated GridWay accessing both the Open Science Grid and EGEE infrastructures. Both of them are accessed using preWS, but offered to the GridWay in the upper level as a WS GRAM interface. For resources on this level, refer to Figure 11.

```
HID PRI OS              ARCH   MHZ %CPU  MEM(F/T)   DISK(F/T) N(U/F/T) LRMS             HOSTNAME
0   1   Linux2.4.21-32. i686  2665  189  964/2006 62787/73964  0/10/10 jobmanager-condor cmsgrid01.hep.wisc.edu
1   1   ScientificSL4   i686   866    0   513/513         0/0  0/16/22 jobmanager-lcgpbs ramses.dsic.upv.es
```

Figure 11: Resources for Gateway Scenario lower level, as seen by `gwhost` in *cepheus*

With this setup, we achieve a common interfaces approach for the upper level, where both TeraGrid and the local UCM cluster are accessed through the same WS interfaces (albeit with different configurations and versions for each of them). Also, at this same level, a GridGateWay is accessed using the same WS interfaces. At the lower level we find that the encapsulated infrastructures are both accessed using different adapters for preWS interfaces than those of upper level. In this way, we are showing the three forms of interoperation in the same scenario: common interfaces in each level, gateways to join both levels and adapters to address the two sets of common interfaces.

Figure 12 provides a visual representation of job distribution in the two levels across the three iterations. The infrastructure labeled as *GGW* is really the server running the GridGateWay, so all its jobs are really being forwarded to one of *EGEE* or *OSG preWS* resources. Therefore middle column of the figure offers a compound view of the *OSG preWS* and *EGEE* resources that the gateway encapsulates. We can still see the expected short amount of jobs taken by the *EGEE* infrastructure and an almost even distribution among the other three. We draw attention the slight increase in jobs taken by both *UCM* and *TG WS* resources with respect to the adapters scenario, since the overhead caused by the GridGateWay increased the jobs diverted to the two upper level infrastructures.

Average suspension times can be seen in Table 3. The *OSG preWS* resource shows a much lower lower suspension time than in the adapters scenario, and it can be seen how this affects to the amount of jobs taken by this infrastructure between iterations. This can be read as less jobs completed whenever they have to wait more in the queue. The other three resources show much more even values, whether comparing between iterations or even between scenarios.

Table 4 shows the productivity for the resources in this scenario. It is a good reflection of the effect caused by the overhead of the gateway, since we can see a slight decrease in the productivity of the encapsulated *OSG preWS* resource, if we compare it with the productivity measured in the previous scenario, even considering the notable decrease in the suspension time. The

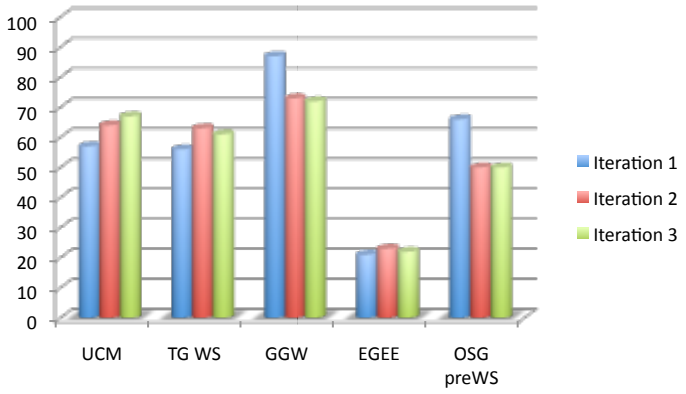

Figure 12: Job distribution across infrastructures: Gateway Scenario

Iteration #	UCM	TG WS	EGEE	OSG preWS
1	220.28	18.73	84.85	20.62
2	215.09	17.76	82.69	39.2
3	201.44	16.32	88.95	38.26

Table 3: Average suspension times as experienced by jobs from the Gateway Scenario

Iteration #	Total	UCM	TG WS	EGEE	OSG preWS
1	132.45	37.75	37.08	13.90	43.70
2	126.58	40.51	39.87	14.55	31.64
3	121.21	40.61	36.97	13.33	30.30

Table 4: Productivity achieved (total and per resource) in the Gateway Scenario

EGEE resource apparently doesn't get affected by this overhead, since its productivity is similar across both scenarios. The most likely explanation is that the reduced performance imposes a stronger condition in the productivity so that the overhead is not shown. The decrease in productivity of the encapsulated resources shows the overhead caused by the gateway and hence the penalty imposed by the use of a hierarchical structure in federation. There is not an overall decrease in productivity due to the better behavior of the *UCM* and *TG WS* resouces, which are directly accessed.

6. Conclusions and Future Work

With this work we have shown that a metascheduler can be used as a tool for federation. Although of course it doesn't solve all the problems alone, it helps to model a solution, becoming an important component to achieve the desired federation. It was argued in this paper that interoperability is not a sufficient condition to produce federation, but other issues have to be taken into account. One of the key conditions of the federation solution has to be its scalability.

We claim that the GridWay metascheduler is a valuable tool for building different types of grid infrastructures. In order to show that our proposed solutions are fit for federation and therefore to sustain our claim, scalability of the metascheduler had to be tested. As a conclusion of that study, GridWay showed great stability, robustness and scalability during this test; where it scheduled 10,000 jobs and kept track of them, showing remarkable robustness and responsiveness. Perhaps the best indicator of this assertion is the fact that it managed to admit 10,000 jobs in pending state in less than four minutes. However, there is work to be done in the transfer MAD, to prevent the memory leaks observed in the scalability test.

As future work in this direction, we want to keep GridWay's value as a tool for federation. Hence, there are plans to develop new MADs for other

infrastructures, more concretely, MADs for UNICORE and for CREAM (the new gLite lightweight service for resource management) are already being developed while also plans for new and more complete scalability and robustness tests are being prepared. New policies for job scheduling are also areas to be explored, to take into account the different topologies of federation described in this paper. In a similar line, the ability to negotiate SLAs in order to automatically acquire resources from an infrastructure provider is being explored, and integration with existing SLA component is currently being study. Another field where is room for improvement is the GridGateWay, for example by reducing its latency to minimize the overhead caused and to be able to improve the productivity achieved by the resources being encapsulated by this component.

Acknowledgements

This work makes use of results produced by the Enabling Grids for EsciencE project, a project co-funded by the European Commission (under contract number INFSO-RI-222667) through the Seventh Framework Programme. EGEE provides a seamless Grid infrastructure available to the European research community 24 hours a day. Full information is available at http://www.eu-egee.org.

Also, we are grateful for resources lent by both the TeraGrid and the Open Science Grid for this experiment.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, The International Journal of High Performance Computing Applications 15 (3) (2001) 200–222.

[2] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers Inc., 1999.

[3] I. Foster, What is the Grid? A Three Point Checklist, GRIDToday 1 (6).

[4] C. Smith, T. Kielmann, S. Newhouse, M. Humphrey, The HPC Basic Profile and SAGA: Standardizing Compute Grid Access in the Open Grid Forum, Concurrency and Computation: Practice and Experience.

[5] M. Riedel, et al., Interoperation of World-Wide Production e-Science Infrastructures, Concurrency and Computation: Practice and Experience.

[6] D. Breuer, P. Wieder, S. van den Berghe, G. von Laszewski, J. MacLaren, D. Nicole, H. C. Hoppe, A UNICORE-Globus Interoperability Layer, Computing and Informatics 21 (2002) 399–411.

[7] P. Kacsuk, T. Kiss, G. Sipos, Solving the Grid Interoperability Problem by P-GRADE Portal at Workflow Level, Future Generation Computer Systems 24 (7) (2008) 744–751.

[8] M. D. Assunção, R. Buyya, S. Venugopal, Intergrid: A Case for Internetworking Islands of Grids, Concurrency and Computation: Practice and Experience 20 (8) (2008) 997–1024.

[9] A. Agarwal, et al., GridX1: A Canadian computational grid, Future Generation Computing Systems 23 (5) (2007) 680–687.

[10] A. Kertesz, P. Kacsuk, Grid Meta-Broker Architecture: Towards an Interoperable Grid Resource, in: CoreGRID Workshop on Grid Middleware in Conjunction with Euro-Par 2006, 2006.

[11] N. Bobroff, L. Fong, S. Kalayci, Y. Liu, J. C. Martinez, I. Rodero, S. M. Sadjadi, D. Villegas, Enabling Interoperability among Meta-Schedulers, in: Proceedings of 8th IEEE International Symposium on Cluster Computing and the Grid (CCGrid-2008), 2008, pp. 306–315.

[12] E. Huedo, R. S. Montero, I. M. Llorente, A Framework for Adaptive Execution on Grids, Software - Practice and Experience 34 (7) (2004) 631–651.

[13] L. Field, Getting Grids to work together, CERN Computer Newsletter 41 (5) (2006) 8–9.

[14] I. Sfiligoi, B. Holzman, Workload Management Systems Evaluation and Integration, Tech. rep., Fermilab (March 2007).

[15] M. A. Frumkin, R. F. Van der Wijngaart, NAS Grid Benchmarks: A Tool for Grid Space Exploration, J. Cluster Computing 5 (3) (2002) 247–255.

[16] 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 5 (65) (2005) 763–771.

[17] E. Huedo, R. S. Montero, I. M. Llorente, A Modular Meta-scheduling Architecture for interfacing with pre-WS and WS Grid Resource Management Services, Future Generation Computing Systems 23 (2) 252–261.

[18] C. Vázquez, E. Huedo, R. S. Montero, I. M. Llorente, Evaluation of A Utility Computing Model based on Federation of Grid Infrastructures, 13th International Euro-Par Conference, Lecture Notes in Computer Science (LNCS) Vol. 4641 (2007) 372–381.