# Coordinated Use of Globus Pre-WS and WS Resource Management Services with Grid*W*ay[⋆]

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

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

**Abstract.** The coexistence of different Grid infrastructures and the advent of Grid services based on Web Services opens an interesting debate about the coordinated harnessing of resources based on different middleware implementations and even different Grid service technologies. In this paper, we present the loosely-coupled architecture of Grid*W*ay, which allows the coordinated use of different Grid infrastructures, although based on different Grid middlewares and services, as well as a straightforward resource sharing. This architecture eases the gradual migration from pre-WS Grid services to WS ones, and even, the long-term coexistence of both. We demonstrate its suitability with the evaluation of the coordinated use of two Grid infrastructures: a research testbed based on Globus WS Grid services, and a production testbed based on Globus pre-WS Grid services, as part of the LCG middleware.

## 1 Introduction

Since the late 1990s, we have witnessed an extraordinary development of Grid technologies. Nowadays, different Grid infrastructures are being deployed within the context of growing national and transnational research projects. The majority of the Grid infrastructures are being built on protocols and services provided by the Globus Toolkit (GT) [1], becoming a *de facto* standard in Grid computing.

The coexistence of several projects, each with its own middleware developments, adaptations, extensions and service technologies, give rise to the idea of coordinated harnessing of resources, or contributing the same resource to more than one project. Moreover, the advent of GT4 and the implementations of Grid services as Web Services by following the WSRF (WS-Resource Framework) [2], arises the idea of a gradual migration from pre-WS Grid services to WS ones, and even, the long-term coexistence of both types of services.

Instead of tailoring the core Grid middleware to our needs (since in such case the resulting infrastructure would be application specific), or homogenizing

[⋆] This research was supported by Ministerio de Educación y Ciencia, through the research grant TIC 2003-01321, and by Instituto Nacional de Técnica Aeroespacial "Esteban Terradas" – Centro de Astrobiología. The authors participate in the EGEE project, funded by the European Union under contract IST-2003-508833.

the underlying resources (since in such case the resulting infrastructure would be a highly distributed cluster), we propose to strictly follow an “end-to-end” principle. In fact, Globus architecture follows an hourglass approach, which is indeed an “end-to-end” principle. In an “end-to-end” architecture, clients have access to a wide range of resources provided through a limited and standardized set of protocols and interfaces. And resources provide their capabilities through the same set of protocols and interfaces. In the Grid these are provided by the core Grid middleware: Globus in this case. Just as, in the Internet, they are provided through the TCP/IP set of protocols.

One approach is the development of gateways between different middleware implementations [3,4]. Another approach, more in line with the Grid philosophy, is the development of client tools that can adapt to different middleware implementations. If we consider that nearly all current projects use Globus as basic Grid middleware, it could be possible a shift of functionality from resources to brokers or clients. This would allow to access resources in a standard way, making the task of sharing resources between organizations and projects easier.

The aim of this paper is to present and evaluate a loosely-coupled architecture that allows the simultaneous and coordinated use of both pre-WS and WS GRAM services, as well as other Grid services. The rest of the paper is as follows. Section 2 compares pre-WS Grid services with WS ones. Section 3 introduces the Globus approach for resource management. Section 4 introduces the Grid*W*ay approach for job management. Section 5 shows some experiences and results. Finally, Section 6 ends up with some conclusions.

## 2 From Pre-WS to WS Grid Services

The main reason behind the moving from pre-WS to WS Grid services is that, according to the Grid’s second requirement proposed by Foster [5], a grid must be built using standard, open, general-purpose protocols and interfaces. However, many people is still reluctant to this change because it could bring an important performance loss. 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 one hand, pre-WS Grid services are based on proprietary interfaces (although usually implemented over standard protocols, like HTTP). On the other hand, WS Grid services are based on the WS-Resource Framework (WSRF) [2], a standard specification fully compatible with other Web Services specifications. In fact, WSRF can be viewed as a set of conventions and usage patterns within the context of established Web Services standards, like WS-Addressing.

WSRF defines the WS-Resource construct as a composition of a Web Service and a stateful resource. The Open Grid Services Infrastructure (OGSI) was previously conceived as an extension of Web Services to have stateful WS-Resources [6]. 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. On the contrary, 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) [7] where tools from multiple vendors interoperate through the same set of protocols and interfaces, implemented in different manners.

## 3 The Globus Approach for Resource Management

The Globus Toolkit [1] 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 [8]. Resource management is maybe the most important component for computational grids, although it could be also extended to other non-computational resources. The Globus Resource Allocation Manager (GRAM) [9] is the core of the resource management pillar of the Globus Toolkit.

In pre-WS GRAM (see Figure 1), when a job is submitted, the request is sent to 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 creates 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.

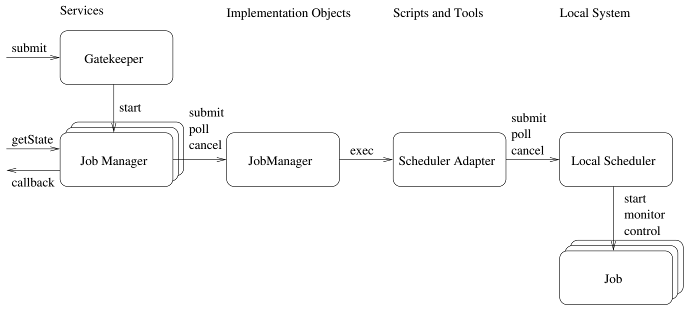

**Fig. 1.** Architecture of the pre-WS Globus Resource Allocation Manager (GRAM)

In WS GRAM (see Figure 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 Services mechanisms and some operations are mapped to a local user via `sudo`. The Managed Job service uses a Job Manager to start and control the job according to its

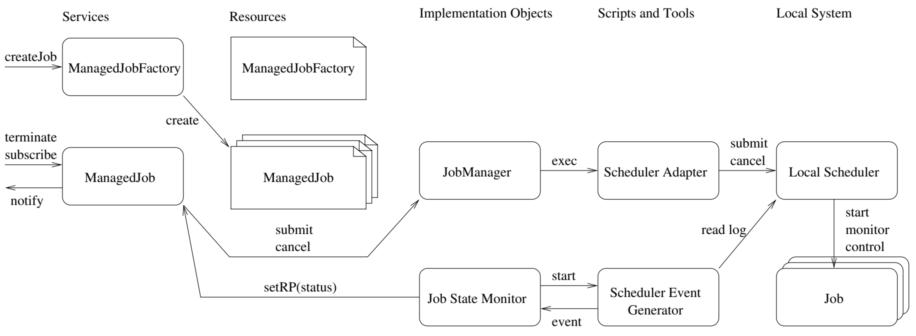

**Fig. 2.** Architecture of the WS Globus Resource Allocation Manager (GRAM)

RSL specification, mapping the request to a local user and communicating state changes back to the GRAM client via WS-Notifications [10]. When the job terminates, either normally or by failing, the Managed Job resource is destroyed, ending the life cycle of the Grid job.

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 GASS (Global Access to Secondary Storage) caching has been removed. In any case, WSRF-based Grid services in GT4 clearly outperforms heavy-weight OGSI-based Grid services in GT3 [11].

As can be seen in Figure 2, WSRF separates services, resources and implementation objects. This way, it 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 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 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.

## 4 The Grid*W*ay Approach for Job Management

Grid*W*ay is a job management system, whose main objective is to provide a decentralized, modular and loosely-coupled architecture for scheduling and executing jobs in dynamic Grid environments. The core of the framework is a personal submission agent that performs all submission stages [8] 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:

- Request Manager (RM): To handle client requests.
- Dispatch Manager (DM): To perform job scheduling.
- Submission Manager (SM): To perform the stages of job execution, including job migration.
- Execution Manager (EM): To execute each job stage.
- Performance Monitor (PM): To evaluate the job performance.

The flexibility of the framework is guaranteed by a well-defined API for each submission agent component. Moreover, the framework has 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 (RS): 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 (MAD): Used by the Execution Manager to submit, monitor and control each job stage.
- Performance Evaluator (PE): Used by the Performance Monitor to check the progress of the job.
- Prolog (P): Used by the Submission Manager to prepare the remote machine and transfer the executable, input and restart (in case of migration) files.
- Wrapper (W): Used by the Submission Manager to run the executable file and capture its exit code.
- Epilog (E): Used by the Submission Manager to transfer back output or restart (in case of stop) files and clean up the remote machine.

Therefore, RS interfaces Grid Information services (e.g. Globus pre-WS and WS MDS), MAD interfaces Resource Management services (e.g. Globus pre-WS and WS GRAM), Prolog and Epilog interfaces Data Management services (e.g. Globus GridFTP, Reliable File Transfer and Data Replication Service), Wrapper interfaces Execution services and PE interfaces Performance services. The result is that the Grid*W*ay core is independent of the underlying middleware.

### 4.1 The Request Manager and Dispatch Manager

The client application uses the Grid*W*ay client API or the DRMAA API [12] 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. 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.

### 4.2 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:

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

This way, 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.

### 4.3 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, monitor and control the execution of the Prolog, Wrapper and Epilog modules. The MAD module provides basic operations with the resource management middleware, like submitting, polling or cancelling jobs, and receives asynchronous notifications about the state of each submitted job. The use of standard input/output makes easy the debugging process of new MADs.

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

## 5 Experiences

### 5.1 Application

In this work we have used the NGB Embarrassingly Distributed (ED) benchmark [13]. The ED benchmark represents the important class of Grid applications called Parameter Sweep Applications (PSA), which constitute multiple independent runs of the same program with different input parameters. This kind of computations appears in many scientific fields like Biology, Pharmacy, or Computational Fluid Dynamics. In spite of the relatively simple structure of these applications, its efficient execution on Grids involves challenging issues [14].

The ED benchmark comprises the execution of several independent tasks. Each one consists in the execution of the SP flow solver [15] with a different initialization parameter for the flow field. In the present work, we have used the FORTRAN serial version of the SP flow solver code. We have used a problem size of class A but, instead of submitting 9 tasks, as NGB class A specifies, we submitted more tasks in order to have a real high-throughput application.

For these experiments 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). Resources are used in a round-robin fashion, as long as they have free slots.

### 5.2 Testbed

In this section, we show the coordinated use of a research testbed with WS GRAM (described in Table 1) and a production testbed (described in Table 2), which is composed of some spanish sites enroled in EGEE, with pre-WS GRAM as part of the LCG (LHC Computing Grid) middleware. The whole testbed is connected by the Spanish National Research and Education Network (RedIRIS).

The resulting environment is highly dynamic and heterogeneous, due to the shared use of compute and network resources, the different DRMS, processors and network links, the different middleware and services, etc. In this case, we have submitted an array jobs with 100 tasks. We have imposed the limitation to only use four nodes simultaneously on each compute resource. In the following experiments, cygnus is used as client.

**Table 1.** Characteristics of the resources in the research testbed

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

**Table 2.** Characteristics of the resources in the production testbed

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

There are several differences between the version of Globus included in LCG and a Globus version installed out of the box. For example, the automatic generation of Grid map files, the use of GLUE schema for MDS, the use of BDII instead of GIIS, and the fact th at file systems are not shared by default between cluster nodes. In a previous work [16], we have shown the coordinated use of two Grid infrastructures, one based on Globus pre-WS services and one based on the LCG middleware, by only using 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 coordinated use of both pre-WS and WS Grid services.

## 5.3 Results

Figures 3 and 4 respectively show the dynamic throughput achieved and scheduling performed during the four experiments. Experiment 1 reaches the maximum throughput (212 jobs/hour) since all resources are available. During experiment

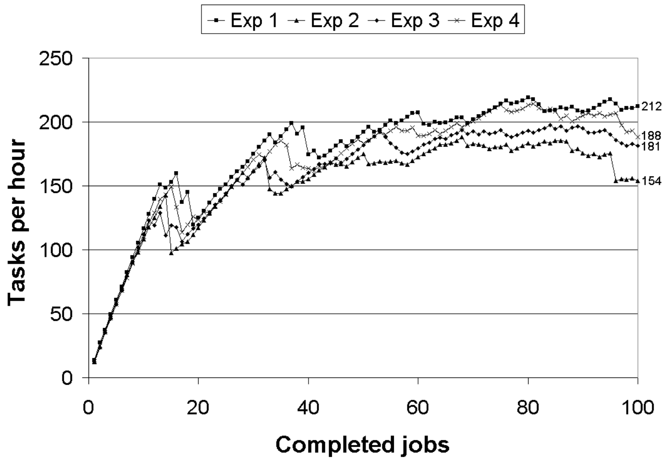

**Fig. 3.** Dynamic throughput in the four experiments

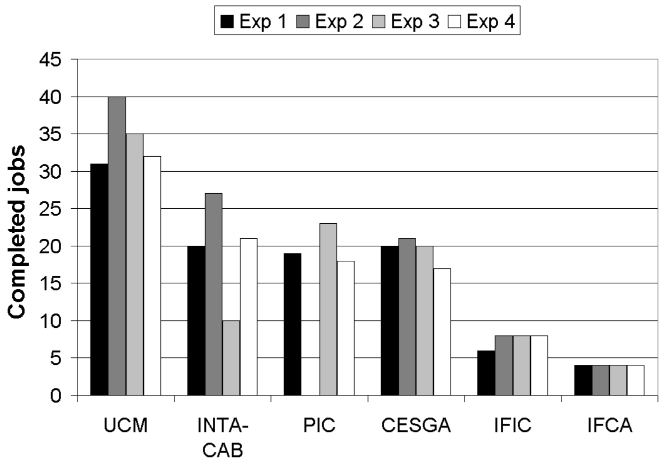

**Fig. 4.** Scheduling performed in the four experiments

2, PIC is unavailable, so no job is allocated to this site and the other sites receive more jobs. Therefore, the throughput drops considerably (154 jobs/hour). During experiment 3, INTA-CAB is partially busy, being only two nodes available for execution. This is reflected in the schedule (INTA-CAB receives half as jobs as in the first experiment) and in the achieved throughput (181 jobs/hour). During experiment 3, CESGA and PIC receive some Grid jobs not related to the experiment. In all the experiments, UCM receives more jobs than the other sites since it presents more compute nodes (10 vs. 4) due to the limitation of four simultaneously running jobs on the same resource.

In most experiments, throughput drops at the end (last five jobs). This is due to bad scheduling decisions (remember the simple RS used) or unexpected conditions triggering job migrations. If there are lots jobs and the testbed is saturated, their effects are hidden, but when few jobs remain, they arise.

## 6 Conclusions

We have shown that our proposed user-level Grid middleware, Grid*W*ay, can work over different Grid infrastructures and service technologies in a *loosely-coupled* way. In this case, we have shown the use of Grid*W*ay over a research testbed based on Globus WS Grid services and a production testbed based on Globus pre-WS Grid services, as part of the LCG middleware, demonstrating that Grid*W*ay can simultaneously work with both pre-WS and WS GRAM. The smooth process of integration of two so different infrastructures and services demonstrates that the Grid*W*ay approach, based on a modular, decentralized and "end-to-end" architecture, is appropriate for the Grid.

We would like to acknowledge all the institutions that have contributed resources to perform the experiments.

## References

1. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. J. Supercomputer Applications **11** (1997) 115–128
2. Czajkowski, K., Ferguson, D.F., Foster, I., et al.: The WS-Resource Framework Version 1.0. Technical report (2004) Available at http://www.globus.org/wsrf.
3. Allan, R.J., Gordon, J., McNab, A., Newhouse, S., Parker, M.: Building Overlapping Grids. Technical report, University of Cambridge (2003)
4. Snelling, D., van den Berghe, S., von Laszewski, G., et al.: A UNICORE Globus Interoperability Layer. Computing and Informatics (2002) 399–411
5. Foster, I.: What Is the Grid? A Three Point Checklist. GRIDtoday **1** (2002) Available at http://www.gridtoday.com/02/0722/100136.html.
6. Foster, I., Czajkowski, K., et al.: Modeling and Managing State in Distributed Systems: The Role of OGSI and WSRF. Proc. IEEE **93** (2005) 604–612
7. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Technical report, OGSI Working Group – GGF (2002)
8. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report GFD-I.4, Scheduling Working Group – GGF (2001)
9. Czajkowski, K., Foster, I., Karonis, N., et al.: A Resource Management Architecture for Metacomputing Systems. In: Proc. IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing. Volume 1459 of LNCS. (1998) 62–82
10. Graham, S., Niblett, P., et al.: Publish-Subscribe Notification for Web Services Version 1.0. Technical report (2004) Available at http://www.globus.org/wsrf.
11. Raicu, I.: 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)
12. Rajic, H., Brobst, R., et al.: Distributed Resource Management Application API Specification 1.0. Technical report, DRMAA Working Group – GGF (2003)
13. Frumkin, M.A., Van der Wijngaart, R.F.: NAS Grid Benchmarks: A Tool for Grid Space Exploration. J. Cluster Computing **5** (2002) 247–255
14. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Adaptive Grid Scheduling of Parameter Sweep Applications. In: Proc. 12th Euromicro Conf. Parallel, Distributed and Network-based Processing (PDP), IEEE CS (2004) 28–33
15. Bailey, D.H., Barszcz, E., Barton, J.T.: The NAS Parallel Benchmarks. J. Supercomputer Applications **5** (1991) 63–73
16. Vázquez, J.L., Huedo, E., Montero, R.S., Llorente, I.M.: Execution of a Bioinformatics Application in a Joint IRISGrid/EGEE Testbed. In: Proc. PPAM Workshop on Large Scale Computations on Grids (LaSCoG). LNCS (2005) (to appear).