EGEE-PUB-2006-036

# gLite Workload Management System Performance Measurements

Svraka, N (IPB) *et al*

Revised on 02 February 2007

Proceedings of IV INDEL, Banjaluka


EGEE is a project funded by the European Commission
Contract number INFSO-RI-508833

The electronic version of this EGEE Publications is available
on the CERN Document Server at the following URL:
<http://cdsweb.cern.ch/search.py?p=EGEE-PUB-2006-036>

EGEE-PUB-2006-036

# GLITE WORKLOAD MANAGEMENT SYSTEM PERFORMANCE MEASUREMENTS

Neda Švraka, Antun Balaž, Aleksandar Belić, Aleksandar Bogojević

*Scientific Computing Laboratory, Institute of Physics, Belgrade, Serbia*

**Abstract –** In this paper an introduction to the gLite Grid middleware and one of its most important components, Workload Management System (WMS), responsible for management of user jobs is given.Useful performance metrics of gLite WMS are defined from a Grid application point of view, and preliminary results of performance measurements are presented and briefly analyzed.

## 1. INTRODUCTION TO GRIDS

Many science experiments generate enormous amounts of data. The processing of this data requires huge computational and storage resources and associated human resources for operation and support. Scientists also face problems requiring vast computing power, i.e. number crunching problems. We can roughly cathegorise these tasks into: tasks with large amounts of distributed data; number crunching tasks; tasks which require simultaneous work of a group of researchers/developers, accessing the same resources at the same time. Please note that typical problems may consist of overlapping tasks from different identified categories, i.e. they may contain computing-intensive analysis of a large amount of distributed data etc. Often a single computer, a cluster of computers or even a special-purpose supercomputer, is not enough for solving challenging science or development problems today.

In order to avoid these obstacles, middleware concept is introduced – layer of software that is able to interconnect distributed computing and storage resources, and make them interoperate, providing users with the unified access to all resources, even if the underlying software (e.g. batch system on individual clusters) or hardware (e.g. different types of storage elements, ranging from tape robots to generic PCs with several HDDs attached) is different. Of course, this middleware layer is built on top of the existing network infrastructure, which is essential for the proper functioning of Grids.

This approach is in some way similar to the World Wide Web (WWW), and people expect that what WWW has done for the information exchange and sharing, the Grids will do for computing resources sharing. However, there are some substantial differences between WWW and Grids: while on the Internet the basic idea is to provide information and we usually have client-server interaction, in Grids the resources are valuable assets and their use should be governed according to the policies of resource providers. In addition, in order to have most efficient use of computing resources available, complex algorithms and internal information system need to be developed and deployed, and a set of new services that will allow simple usage by the end users provided.

There are many kinds of Grids with different purposes, such as national Grid infrastructures (aiming to couple high-end resources across a nation, e.g. AEGIS [1] in Serbia, or the UK e-Science program), project Grids (funded by certain funding agencies, goodwill Grid infrastructures provided by individuals aiming to help in solving important common problems (e.g. in finding drugs for diseases), consumer Grids established by commercial companies, etc.

Project Grids are currently the main providers of different middleware distributions, some of which are freely available, thus enabling general public to join the Grid, or to adapt it for their own needs. Project Grids are created to meet the needs of a variety of multi-institutional research groups and multi-company "virtual teams", to pursue short- or medium-term projects (scientific collaborations, engineering projects). Such a project is World Wide LHC Computing Grid Project (WLCG)[3], which was created to prepare the computing infrastructure for the simulation, processing and analysis of the data of the Large Hadron Collider (LHC) experiments. The LHC, which is being constructed at the European Laboratory for Particle Physics (CERN), will be the world's largest and most powerful particle accelerator.

The WLCG project shares a large part of its infrastructure and works in conjunction with the Enabling Grids for E-Science (EGEE-II) project [4], large European E-infrastructure project with the main goal is to provide researchers with access to a geographically distributed computing Grid infrastructure, available 24 hours a day. SEE-GRID-2 [5] is the regional project aiming to provide Grid infrastructure in the South East Europe region, incubate new regional communities, and stimulate development of new Grid-aware applications.

## 2. INTRODUCTION TO MIDDLEWARE

The essence of the Grid is the software that enables the user to access computers distributed over the network. This software is called "middleware", because it is distinct from the operating systems software that makes the computers run (e.g. Linux) and also different from the applications software that solves a particular problem for a user (e.g. a computer visualization programme). The term "middleware" reffers to the fact that it is conceptually in between these two types of software.

The middleware's task is to organize and integrate the distributed computational resources of the Grid into a coherent structure. This means the objective of the middleware is to get the applications to run on the right computers, wherever they may be on the Grid, in an efficient and reliable way. It also provides users with a single interface to the Grid.

Different distributions of middleware exist today – Globus, LCG, gLite, UNICORE, GAT. The gLite [6] is successor of the LCG-2 middleware, and is most widely used.

The EGEE-II project focuses on maintaining the gLite middleware and on operating a large computing

infrastructure for the benefit of a vast and diverse research community. The gLite middleware hides much of the complexity of this environment from the user, giving the impression that all of these resources are available in a coherent virtual computer centre.

We will now in brief describe basic entities ("building blocks") and available interfaces which allow user to run jobs and manage data [7].

The access point to the WLCG/EGEE-II/SEE-GRID-2 Grid is the User Interface (UI). This can be any machine where users have a personal account and where their user digital certificate is installed. From a UI, a user can be authenticated and authorized to use the WLCG/EGEE/SEE-GRID-2 resources, and can access the functionalities offered by the Information, Workload and Data management systems.

A Computing Element (CE) is a set of computing resources localized at a site (often referred to as a cluster, or a computing farm).

A Storage Element (SE) provides uniform access to storage resources at a certain site. The Storage Element may control simple disk servers, large disk arrays or tape-based Mass Storage Systems (MSS). Most WLCG/EGEE/SEE-GRID-2 sites provide at least one SE. Storage Elements can support different data access protocols and interfaces.

The Information Service (IS) provides information about the Grid resources and their status.

In a Grid environment, files can have replicas at many different sites. Ideally, the users do not need to know where a file is located, as they use logical names for the files that the Data Management services will use to locate and access them.

The Workload Management System (WMS) [4] accepts user jobs, assigns them to the most appropriate Computing Element, records their status and retrieve their output. The Resource Broker (RB) is the machine where the WMS services run.

Finally, the Logging and Bookkeeping service (LB) tracks jobs managed by the WMS. It collects events from many WMS components and records the status and history of the job.

## 3. HOW DOES THE WMS WORK?

This paper is devoted to the measurement of the performance of the WMS [8]. As mentioned before, the purpose of WMS is to accept requests for job submission and management coming from its clients and take the appropriate actions to satisfy them. The complexity of the management of applications and resources in the grid is hidden by the WMS to the users. Their interaction with the WMS is limited to the description of the characteristics and requirements of the request via a high-level, user-oriented specification language, the Job Description Language (JDL) and to the submission of it through the provided interfaces. The WMS is responsible for translation these abstract resource requirements into a set of actual resources, taken from the overall grid resource pool, to which the user has access permission.

The JDL allows the description of the following request types supported by the WMS:

• Job: a simple application

• DAG: a direct acyclic graph of dependent jobs

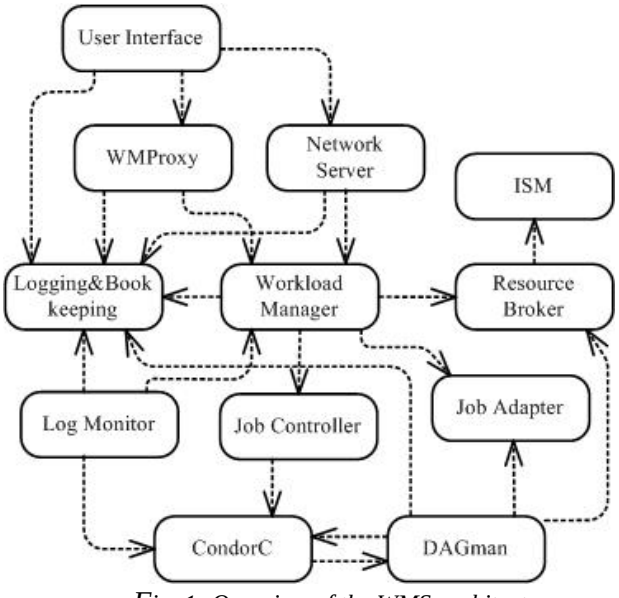
• Collection/Bulk: a set of independent jobs

There is a set of client tools, referred to as WMS-UI, which allows the user to access the main services (job management services). These client tools include a command line interface, a graphical interface and an API, providing both C++ and Java bindings, which allow the requests to be submitted and managed programmatically. Through the WMS UI user can find the list of resources suitable to run a specific job, submit a job/DAG for execution on a remote Computing Element, check the status of a submitted job/DAG, cancel one or more submitted jobs/DAGs, retrieve the output files of a completed job/DAG (output sandbox), retrieve and display logging and bookkeeping information about submitted jobs/DAGs.

After submission, the request passes through several components of the WMS, before it completes its execution. The internal architecture of the WMS is given in Fig. 1. There are two approaches for acceptance of incoming requests, one is based on a generic daemon and the other on the Web Services based interface. These two modules are the key subject of measurements performed in this paper.

The Network Server (NS) is a generic network daemon that provides support for the job control functionality. It is responsible for accepting incoming requests from the WMS-UI (e.g. job submission, job removal), which, if valid, are then passed to the Workload Manager.

The Workload Manager Proxy (WMProxy) is a service providing access to WMS functionality through a Web Services based interface. Besides being the natural replacement of the NS in the passage to the SOA approach for the WMS architecture, it provides additional features such as bulk submission and the support for shared and compressed sandboxes for compound jobs.




*Fig, 1: Overview of the WMS architecture.*

The Workload Manager (WM) is the core component of the Workload Management System. Given a valid request, it has to take the appropriate actions to satisfy it. It coordinates other modules that provide a matchmaking

service (Resource Broker), the actual job management operations (CondorC), preparation of the CondorC submission file and creation the appropriate execution environment in the CE worker node (Job Adapter).

The Logging and Bookkeeping (LB) service provides support for job monitoring functionality: it stores all information concerning events generated by the various components of the WMS.

For a generic job there are two main types of request: **submission** and **cancellation**. The submission request passes the responsibility of the job to the WM. The WM will then pass the job to an appropriate CE for execution, taking into account the requirements and the job preferences expressed in the job description file. The decision on which resource is to be used is the outcome of the matchmaking process between the submission requests and the available resources. The job can also be cancelled by the user at any time after it is submitted using the job ID that uniquely identifies each job.

## 4. WMS PERFORMANCE

In order to assess performance of the WMS, especially the process of submitting a long series of jobs (which is a typical use-case scenario for an application that requires vast computing resources and is for this reason ported to the Grid), we developed a series of WMS tests. In our test environment long series of jobs with different requirements have been submitted and timing of critical job events has been recorded and analyzed.

The testbed environment included a single User Interface, and a single WMS collocated with a top-level BDII, which provides database on available resources, used in the matchmaking process by WMS. User Interface was a laptop machine (Pentium M, 1.8 GHz, 512 MB RAM, 100 Mbps network card), while the WMS/BDII node was double Xeon 2.8 GHz with hyperthreading enabled, 2 GB of RAM, 1 Gbps network card. Both machines were connected to the same high-quality 3Com Gigabit network switch. The latest gLite 3.0.2 middleware was installed on both nodes.

In the first series of tests, jobs have been sent via a Network Server, and in the second one via Workload Manager Proxy. Information associated with each job status was obtained from Logging and Bookkeeping service for both cases. The Logging and Bookkeeping service is collocated with the WMS service. We were interested to find out how the typical submission time per job changes with the change of type of submission: sequential (thread) submission of jobs to both NS and WMProxy, as well as for bulk submission to WMProxy. We also investigated if changing the overall number of submitted jobs will influence the frequency of submission, and the dependence of the submission frequency on the size of job Input Sandboxes (files associated with each job that need to be uploaded to the WMS during the job submission). The client performs action running scripts based on the Command Line Interface (CLI) commands from the User Interface.

For the first measurement, client instantiates a number of threads and each thread executes sequentially a given number of job submission commands. The jobs were just self contained JDLs (no sandboxes). Numbers of jobs used in such submissions were 100, 500 and 1000. The second type of measurements assumes the same approach, but jobs were described with JDLs containing small Input Sandboxes, with the size of approximately 8 kB.

Also, it was interesting to examine a new feature, introduced by WMProxy, bulk submission of jobs, i.e. parallel submission of a collection of jobs using a single command line. Tests were performed with different number of jobs in collection (100, 500 and 1000) and different size of sandbox (no sandbox, as well as a sandbox of 8 kB). Results of measurements are shown in Fig. 1.

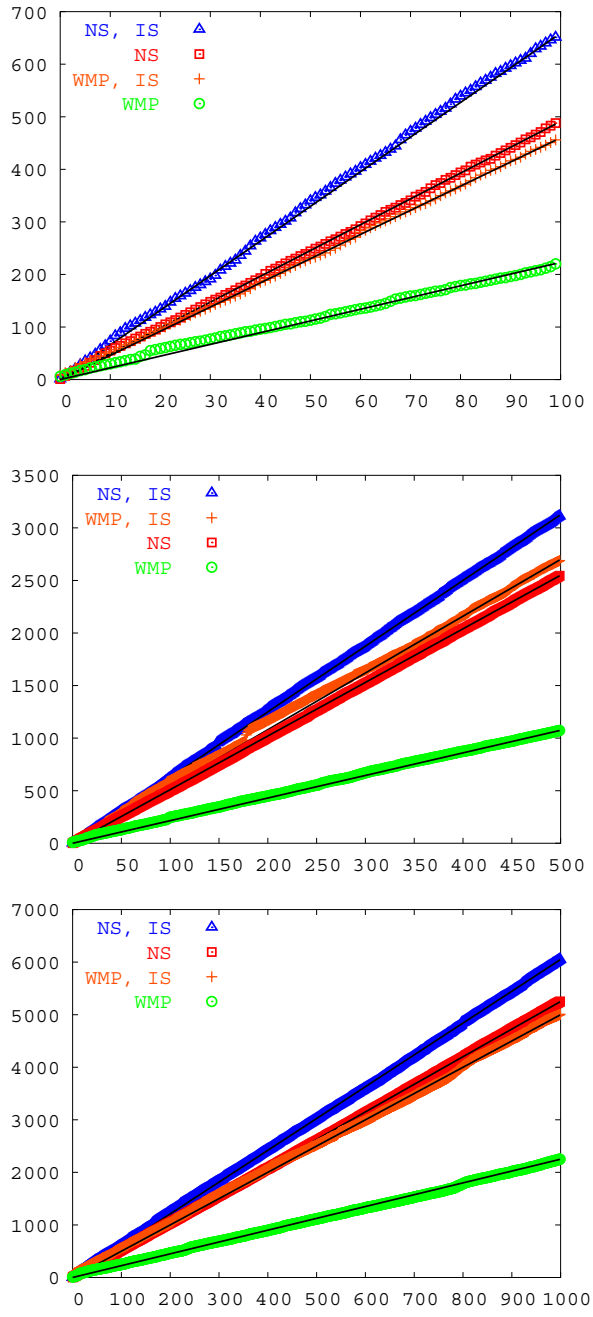



*Fig.2: Time (in seconds, on y-axis) needed for a submission of a large number of jobs (on x-axis) for 100, 500, and 1000 jobs. The jobs were submitted through the Network Server interface, without (NS) and with a small Input Sandbox (NS, IS), and through the WMProxy interface without (WMP) and with a small Input Sandbox (WMP, IS).*

The three graphs in Fig. 2 represent the dependence of the submission time on the number of jobs. The overall number of submitted jobs is 100 on the top plot, 500 on the mid one, and 1000 on the bottom one. Comparing the performance of Network Server and WMProxy, we see that WMProxy outperforms the corresponding Network Server measurements in both cases considered (no sandbox, small sandbox). The fact that each of these curves is actually a linear function shows that there is no saturation in WMS performance, and that it can accept large number of jobs without having its performance reduced. The slope of each curve in Fig. 2 represents typical submission time per job.

Therefore, we see that the usage of WMProxy consumes much less time for submission of a single job then the usage of Network Server. For the thread of 100 jobs the submission with WMProxy takes about 2.2 seconds per job with no sandbox, and about 4.5 seconds per job with small sandbox. On the other hand, Network Server needs 4.9 seconds in the first case, and 6.5 seconds in the second one. We also see that the presence of even a small sandbox affects performance of WMProxy service drastically (two times longer submission time), while the increase in the submission time is not so prominent with the Network Server (1.3 longer submission time). The submission of longer threads of jobs (500, 1000) does not give substantially different average job submission times.

The other interesting quantity we investigated is the average frequency with which jobs can be submitted using either service. This is the inverse value of slopes for Fig. 2. This way we can compare performance of NS and WMProxy services with the performance of a bulk (collection) jobs submission. The results are presented in Fig. 3.

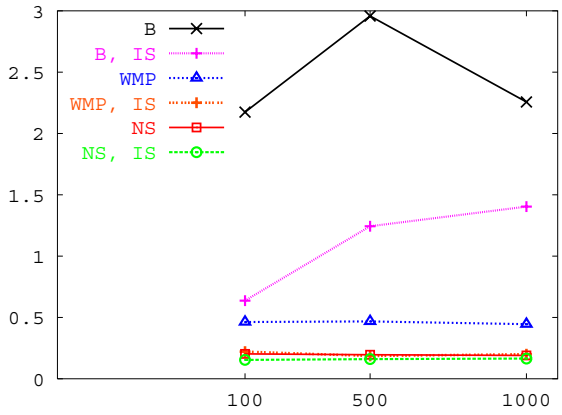



*Fig. 3: The average job submission frequency (on the y-axis) achieved during submission of different numbers of jobs (on the x-axis). The jobs were submitted through the Network Server interface, without (NS) and with a small Input Sandbox (NS, IS), and through the WMProxy interface without (WMP) and with a small Input Sandbox (WMP, IS), as well as using the Bulk submission without (B) and with a small Input Sandbox (B, IS).*

While the average frequency of non-bulk submission ranges from approximately 0.20 jobs per second (no sandbox) to 0.16 jobs per second (small sandbox) for NS, or 0.46 jobs per second (no sandbox) to 0.20 jobs per second (small sandbox) for WMProxy, the bulk submission has much better performance. As we see from Fig. 3, bulk submission frequency ranges from approximately 2.5 jobs per second (no sandbox) to around 1 job per second (small sandbox).

The performed measurements represent just the preliminary results, and we are planning to do a more complex investigation of WMS performance and stability, such as parallel submission of threads of jobs from two or more User Interfaces, transferring large Input Sandboxes (~MB), etc. Insights gained from such measurements can be very useful not only to the middleware developers aiming to improve the performance of Grid services, but also to the most important group of people – Grid users – which must take into account these results when planning gridification of their applications. Such knowledge enable them to choose the most efficient approach for porting applications to Grids.

## 5. CONCLUSIONS

We presented preliminary results of gLite Workload Management System performance measurements. For job thread of different sizes (100, 500, 1000) we measured the average submission time per job and frequency of job submissions for Network Server, WMProxy, and bulk submission. We found that WMProxy outperforms Network Server service in all considered cases (2 to 1.5 times, depending on the size of sandbox), with WMProxy performance being more sensitive to the size of the sandbox. We also found that the bulk submission of jobs is far superior service, giving consistently 10 times faster response than the NS, and 5 times faster response than the WMProxy.

## 6. ACKNOWLEDGMENTS

This work was supported in part by the Ministry of Science and Environmental Protection of the Republic of Serbia under project no. OI141035. The presented numerical results were obtained on the AEGIS GRID e-infrastructure[5] whose operation is supported in part by EC FP6 projects EGEE-II (INFSO-RI-031688) and SEE-GRID-2 (INFSO-RI-031775).

## 7. REFERENCES

[1] http://aegis.phy.bg.ac.yu/
[2] http://lcg.web.cern.ch/LCG/
[3] http://www.eu-egee.org/
[4] http://www.see-grid.eu/
[5] http://glite.web.cern.ch/
[6] gLite 3.0 User Guide, https://edms.cern.ch/document/722398/1/
[7] WMS Guide, https://edms.cern.ch/document/572489/1

**Сажетак –** Описан је gLite Grid middleware и једна од његових најважнијих компоненти - Workload Management System (WMS), одговорна за управљање корисничким пословима и подацима. Приказани су и укратко анализирани прелиминарни резултати мерења перформанси WMS-а, дефинисани са тачке гледишта оптимизовања Grid апликације

**МЕРЕЊЕ ПЕРФОРМАНСИ GLITE WORKLOAD MANAGEMENT СИСТЕМА**

Неда Шврака, Антун Балаж,
Александар Белић, Александар Богојевић