The gLite Workload Management System

Abstract. *The gLite Workload Management System (WMS) is a collection of components that provide the service responsible for distributing and managing tasks across computing and storage resources available on a Grid. The WMS basically receives requests of job execution from a client, finds the required appropriate resources, then dispatches and follows the jobs until completion, handling failure whenever possible. Other than single batch-like jobs, compound job types handled by the WMS are Directed Acyclic Graphs (a set of jobs where the input/output/execution of one of more jobs may depend on one or more other jobs), Parametric Jobs (multiple jobs with one parametrized description), and Collections (multiple jobs with a common description). Jobs are described via a flexible, high-level Job Definition Language (JDL). New functionality was recently added to the system (use of Service Discovery for obtaining new service endpoints to be contacted, automatic sandbox files archival/compression and sharing, support for bulk-submission and bulk-matchmaking). Intensive testing and troubleshooting allowed to dramatically increase both job submission rate and service stability. Future developments of the gLite WMS will be focused on reducing external software dependency, improving portability, robustness and usability.*

ANDREETTO, Paolo (INFN); ANDREOZZI, Sergio (INFN); AVELLINO, Giuseppe (DATAMAT); BECO, Stefano (DATAMAT); CAVALLINI, Andrea (DATAMAT); CECCHI, Marco (INFN); CIASCHINI, Vincenzo (INFN); DORISE, Alvise (INFN); GIACOMINI, Francesco (INFN); GIANELLE, Alessio (INFN); GRANDINETTI, Ugo (DATAMAT); GUARISE, Andrea (INFN); KROP, Andrea (DATAMAT); LOPS, Roberto (INFN); MARASCHINI, Alessandro (DATAMAT); MARTELLI, Vincenzo (INFN); MARZOLLA, Moreno (INFN); MEZZADRI, Marco (INFN); MOLINARI, Elisabetta (INFN); MONFORTE, Salvatore (INFN); PACINI, Fabrizio (DATAMAT); PAPPALARDO, Marco (INFN); PARRINI, Andrea (DATAMAT); PATANIA, Giuseppe (INFN); PETRONZIO, Luca (DATAMAT); PIRO, Rosario (INFN); PORCIANI, Maurizio (DATAMAT); PRELZ, Francesco (INFN); REBATTO, David (INFN); RONCHIERI, Elisabetta (INFN); SGARAVATTO, Massimo (INFN); VENTURI, Valerio (INFN); ZANGRANDO, Luigi (INFN)

1. Introduction
Scheduling of distributed, data-driven applications in a Grid environment is a challenging problem in the diverse domains of workload management, resource discovery, matchmaking and brokering, accounting, authorization policies, resource access, reliability and dependability. Although significant results were achieved in the past few years, the development and the proper deployment of generic, robust, reliable and standard components operating on worldwide scale, has brought out non trivial issues requiring joint efforts with a strong degree of cooperation to

be worked out.
All these issues are currently being tackled within the EU-funded EGEE project (Enabling Grids for E-science in Europe) [1] - now at its second phase, EGEE-II - whose primary goals are the provision of robust middleware components and the creation of a quality Grid infrastructure to support e-Science applications. It is a large scale project built on previous European projects and on national, regional and thematic initiatives with an extensive programme of middleware re-engineering that has resulted in a consolidated software stack, gLite [2].
The Workload Management System is one of the key Grid services composing the gLite middleware stack. It has been designed with some fundamental principles in mind: first of all aiming to provide a dependable and reliable service, never losing track of jobs to be processed and always providing a prompt, responsive quality of service yet keeping up with huge and even growing factors of scale. It is designed as part of a Service Oriented Architecture (SOA) complying with Web-Service Interoperability (WS-I) [3] specifications and strives to implement recommendations on web service foundations made by the Open Grid Forum (OGF) [4].
Fundamental to any Grid environment is the ability to discover, allocate and monitor the use of resources. The term "workload management" is commonly used to describe the process of discovering resources all over the Grid, selecting the most suitable ones, arranging for submission, monitoring and information gathering. In this respect, the WMS has to deal with a heterogeneous computing scenario that in general encompasses different architectures and loss of centralised control also in presence of potential faults due to the distributed and diverse nature of the Grid environment, computers, networks and storage devices.
In this paper we will show what has been achieved to provide adequate workload and management components, suitable to be deployed in a production-quality Grid, covering the design and development of the gLite WMS with respect to functionality, interoperability and reliability. We also report on achieved results and outline some possible future directions.

2. Functionality

The gLite Workload Management System (WMS) provides a service responsible for the distribution and management of tasks across resources available on a Grid, in such a way that applications are conveniently, efficiently and effectively executed. These tasks, that basically request computation and storage, are usually referred to as "jobs". The WMS supports different types of jobs:

- Single batch jobs
- DAGs: jobs with dependencies expressed as a direct acyclic graph. This is implemented using DAGMan, a meta-scheduler from the Condor [5] suite, whose purpose is to navigate the dependency graph determining the order in which the nodes are to be executed, then following the execution of the corresponding jobs where each instance is inserted into and handled by a Condor queue.
- Collections: sets of independent jobs sharing similar requirements that can be matched in clusters according to some significant attributes and then tracked with a single handle
- MPI: based on message passing interface - a widely-used library to allow parallel programming
- Interactive: establishing a synchronous two way communication with the user on a socket stream
- Parametric: allowing multiple jobs to be defined by a single description with attributes varying with a parameter

The characteristics of a job are defined using a flexible and expressive formalism called Job Description Language (JDL) [6]. The JDL is based on Classified Advertisements or *ClassAds* [7], developed within the Condor project, which consist basically of a list of key/value pairs that represent the various characteristics of a job (input files, arguments, executable, etc.) as well as its requirements, constraints and preferences (memory, operating system, etc.).
Jobs are always associated with user proxy credentials and the job-dependent operations are performed on behalf of the user. gLite in general and the WMS in particular exploit experience and existing components from the VDT (Virtual Data Toolkit from Condor and Globus [8]). Condor plays a significant role in the present architecture as a job submission and tracking layer (see later), while the Globus Security Infrastructure (GSI) is used throughout.
A mechanism also exists to renew credentials automatically and safely for long-running jobs. It is possible in fact that long jobs may outlive the validity of the initial proxy; if so and the proxy is not renewed, the job will die prematurely. To avoid this the WMS allows the proxy to be renewed automatically when the user allowed his credentials to be managed by a dedicated service called MyProxy [9].
Submitting a job actually means passing its responsibility to the WMS whose purpose is then to find the appropriate resources matching the requirements, watching and directing the job on its way to completion, with particular attention to infrastructure failures requiring resubmission. The WMS will in the end forward the job to one or more appropriate Computing Elements (CE) for execution. The decision about which resource is adequate to run the job is the outcome of a match-making process between the "demand", represented by the submission requirements and preferences, and the "offer", represented by the characteristics of the available resources. In case a match between a job and one or more suitable CEs is not found, the job is kept pending by the WMS and periodically retried until the request expires.
The Grid is a complex system and things can go wrong at various stages throughout. The WMS has some ability to both prevent and recover from failures. While using persistent data structures helps never losing track of the user's incoming or pending requests, resubmission of failed jobs gives the user a higher degree of reliability about the final outcome of his jobs. Resubmission can be achieved at two levels. "Shallow" resubmission is utilised in those cases where an error occurs before the CE has started executing the job, in which case another CE can be tried immediately without any worry to compromise the results. This will also reduce the probability to have multiple instances of the same job over the Grid due to temporary loss of contact. "Deep" resubmission happens when a job fails after it starts running; this can be more problematic as the job may well have done a considerable amount of processing, producing output files or making other state changes, and may have consumed a significant amount of time. Users can therefore choose the number of times they will allow the job to be resubmitted in these two ways with two JDL parameters.

Besides request submission, the WMS also implements request management and control functionality such as cancellation and output retrieval. Another feature exists to list all the available resources matching a given job at a given time. Request status follow-up can be instead achieved through the Logging & Bookeeping service [10]. The WMS tightly depends on the L&B services.
The various job management tasks mentioned so far are executed by different processes or threads that communicate via persistent data structures [Figure 1]. One core component is the Match-Maker, which sorts out a list of resources suitable for the given requirements. These resources may include Storage Elements (SE) if the user specified the need to manipulate data. The list of the resources that match the specified requirements is sorted in order of decreasing rank, and the highest-ranked resource will in general be chosen. The ranking criteria are provided by the user in the JDL. To compensate the delay in the update of the status of Batch Systems in the Information System (which can cause the uneven distribution of short bursts of jobs, always

being assigned to the top ranked resource), a so called "fuzzy rank" stochastic algorithm can be used to perform a random selection among the matching resources - weighted according to their actual rank - yielding a smoother distribution of the load.
Access to some dataset can be specified as part of the job requirements, as well as the need to have the job running "close" to some specific Storage Element. The WMS also allows to specify requirements on the Storage Elements that provide the requested data via a "gang"-matching function. While matchmaking is made between two parties, typically the job and the computing resource - gang-matching allows to take into account the characteristics of multiple kind of resources (e.g. the SEs as well), as provided by the Information System. In general, more or less "eager" or "lazy" policies can be adopted for scheduling. At one extreme, eager scheduling dictates that a job is bound to a resource as soon as possible and, once the decision has been taken, the job is passed to the selected resource(s) for execution, where, very likely, it will end up in some queue. This mechanism is usually referred to as 'push mode'. At the other extreme, lazy scheduling foresees that the job is held by the WMS until a resource becomes available, at which point that resource is matched against the submitted jobs and the job that fits best is passed to the resource; this is called 'pull mode'.

The implementation feature that allows the flexible application of different policies is the decoupling between the collection of information about resources and its use. The component that implements this mechanism is called Information Supermarket (ISM) which basically consists of a repository of resource information, all residing in memory, whose update is the result of either the arrival of notifications or active polling of resources or some arbitrary combination of both from different source of Information Providers. Moreover, the ISM can be configured so that certain notifications can trigger the matchmaking process itself, thus supporting the implementation of lazy scheduling policies. The ISM represents one of the specific improvements to the WMS as inherited from the EDG [11] (and LCG) projects where the information was collected in real-time, contacting Information Providers for each single request, which is less efficient and reliable.

Reflecting the demand-offer/job-resource symmetry, all the job requests are kept in a structure called 'Task Queue', similarly to what happens for resources, where they queue up and wait to call or to be called for a match. This also allows to hold a submission request if no matching resources are immediately found and provides a necessary mechanism for the support of lazy scheduling policies. Non-matching, waiting requests will be retried either periodically or as soon as notifications of available resources get to the ISM according to the adopted scheduling model. A limiter mechanism based on system vital parameters such as load, disk and memory usage, network activity etc. has also been recently implemented to protect the system from congestion by preventing new requests from being accepted after reaching some configurable thresholds. Experience has shown that always accepting new requests that can make little or no way to progress deteriorates performance inevitably leading to a state of general unresponsiveness of the whole system whose specific trigger is very hard to predict and can vary significantly depending on the nature of the requests being processed: this calls for a feedback mechanism based on system-wide vital statistics.

Here is a summary of non-trivial functionality implemented by the gLite WMS:

- Resubmission: shallow or deep
- Bulk-matchmaking
- Fuzzy ranking
- Support for MPI jobs even if the file system is not shared between CE and Worker Nodes (WN)

- Support for execution of all DAG nodes within a single CE - chosen by either user or by the WMS match-maker
- Support for file peeking to access files during job execution
- Load limiting mechanism to prevent system congestion
- Automatic sandbox files archiving/compression and sharing between jobs
- Gang-matching to include Storage Elements in the match-making process.

3. Interoperability

Given the typically large number of different parties involved in a Grid infrastructure, interoperability plays a key role to facilitate establishing and coordinating agreements and interactions between them. In this respect, the WMS, especially by virtue of his central, mediating role, has to deal with a wide variety of people, services, protocols and more, ranging from users - belonging to different VOs - to other services of the EGEE/gLite infrastructure and to other Grids as well.

For what concerns users, to be able to allow interaction adhering to the SOA model, a Simple Object Access Protocol (SOAP) Web Service has been implemented, its interface being described through a Web Service Description Language (WSDL) specification written in accordance to the WS-I profile. It replaced a legacy network interface based on a proprietary protocol. This Web Service manages user authentication/authorization and operation requests. It runs in an Apache container extended with the FastCGI and GridSite modules. The FastCGI module implements Common Gateway Interface (CGI) functionality along with some other specific features. FastCGI applications are able to serve, in a multiprocessing fashion, multiple requests, where instances can be dynamically spawned or terminated according to the demand. In particular, an additional control mechanism over unpredictable error conditions such as undefinite hanging has been implemented to automatically terminate a serving process of the pool after a given configurable number of requests. Moreover, the Grid Site module provides an extension to the Apache Web Server for use within Grid frameworks by adding support for Grid Security Infrastructure (GSI), the Virtual Organization Membership Service (VOMS) [13] and file transfer over HTTPS. It also provides a library for handling Grid Access Control Lists (GACL).

On the topic of interoperation with other Grid services, it is worth to first describe in more detail how job management is accomplished by the WMS. A service called Job Submission Service (JSS) is responsible to actually forward the job to the chosen resource and monitor its execution. The submission tool to the CE in current use is based on Condor-G and is responsible for performing the actual job management operations. Every CE supported by Condor-G is then implicitly supported by the WMS as well, in particular the LCG CE (pre-WS Condor-G plus GRAM on the CE) and the gLite CE (pre-WS Condor-G plus Condor-C, or just Condor on the CE). A monitoring service, part of the JSS, is also responsible for watching the Condor log files intercepting interesting events concerning active jobs, events affecting the job state machine and triggering appropriate actions. After the recent development of CREAM [14] (a new WS-I/BES [15] compliant CE) a new JSS component, called ICE, has been introduced to peer CREAM and manage jobs. CREAM has a WS-based interface, extension of the Java-Axis servlet running inside an Apache Tomcat container. This allows interoperability through WSDL. Accordingly, ICE is a gSOAP/C++ layer which will securely send job operations to a CREAM CE. In doing so, it subscribes to the CEMon service in order to asynchronously receive notifications about job status changes. In case some notifications are lost, ICE also performs synchronous status polling for unresponsive jobs.

Interoperation with Information Providers can be achieved either syncronously or asyncronously for those providers who support it. We actually provide interfacing with the Berkeley Database Information Index (BDII), CeMon (synchronously and asynchronously) and Relational Grid

Monitoring Architecture (R-GMA). As mentioned earlier, all resource information, however collected, is gathered in the Information Supermarket for caching.
For job definition, the WMS also fully endorses the Job Submission Description Language (JSDL). This is an emerging OGF standard which aims to facilitate interoperability in heterogeneous environments, through the use of an XML based job description language that is free of platform and language bindings. The JSDL language contains a vocabulary and normative XML Schema that facilitate the expression of those requirements as a set of XML elements. While JSDL serves the same purpose and has semantics comparable to the current ClassAd-based JDL, its adoption as an OGF approved standard makes it a good candidate for support by the WMS system.
On the front of interoperabily between Grids, some work has recently been done to be able to integrate the Open Science Grid within the EGEE project. A significant number of OSG resources are now properly seen by the gLite Information Systems thanks to a fruitful and coordinated joint-effort. The first tests of job submission were encouragingly successful. This interoperability requirement has required a further level of abstraction to be implemented in the WMS jobwrapper (the shell script generated by the WMS which surrounds the user job execution and performs basic setup and cleanup operations, downloading/uploading the sandbox, setting the execution environment, logging etc.). Due to the diverse nature of resources belonging to one or more Grids, the jobwrapper script must be kept as simple and as robust as possible. The jobwrapper script may be running in an unfriendly worker node environment where no or little assumption can be made on what is available. Again, due to the pivotal role of this script, a significant work has also been done to extend it in order to encompass all the different requirements expressed by the involved parties (users, VOs and resources) without losing functionality nor generality. To achieve this, a clean set of hooks was provided in the jobwrapper generation procedure, allowing specific customisations to be inserted by users, VO managers and site administrators. This approach reduces hard-coding without limiting functionality. For users, prologue and epilogue scripts have been included - these are run before and after the job is executed - basically with the intent of letting them set and clean up the proper environment for their jobs; for VOs, a hook is present to retrieve the proper middleware version; for resource managers, customization points are disseminated throughout the script.

Here's a summarized view of the functionality provided in the areas of integration with other services and interoperability:

- Backward compatibility with LCG-2
- Automatic renewal of credentials
- GridFTP and HTTPS to handle secure file transfer for the sandbox
- Service Discovery for obtaining new service endpoints to be contacted
- Support of different mechanisms to populate the ISM from several sources (BDII, R-GMA, CeMon)
- Support for submission and monitoring for the LCG, gLite and CREAM CEs
- Support for Data management interfaces (DLI and StorageIndex)
- Support for JSDL
- Interoperability with the american Open Science Grid (OSG)
- Integration with DGAS [12] - a Grid accounting system
- Integration with G-Pbox [13] - a Grid policy framework
- User prologue/epilogue, VO and resource hooks to be run by the jobwrapper in the WN.

4. Reliability and performance

Thorough testing and an intense troubleshooting activity, which required a remarkable degree of collaboration with end users that had access to resource sets of large enough scale, we were able to significantly increase job submission rates and service stability. A new release of the WMS (3.1) is being deployed, as of late 2007, instrumented with all the features and optimizations coming from this recent, intense work. A special mention has to be spent for the introduction of the bulk-submission and matchmaking features. Originally, in fact, all collections were managed trasforming them into a 2-level DAG having dependencies pointing from each leaf to a root node. The way DAGman was used within our architecture was convenient at first but overdesigned, in the end not efficient for handling disjoint jobs with no dependedencies among them. Keeping resource under control in our application of DAGMan also required to apply workarounds (e.g. global limit on the number of planner processes, as DAGman provides no throttles that apply to node pre- and post-script execution). Reimplementing job collection handling to be directly managed by the WMS (sort of generalizing the submission of a single job to the submission of a set of jobs to be matched together in one single shot) has boosted performance and improved reliability. For the sake of clarity, the set of jobs specified by the collection is split into subsets identified by the equivalence classes generated by the specification - explicitly done by the user - of some "significant" attributes whose literal equality identifies equivalent jobs. A typical use-case for specifying significant attributes could be, as an example, parting the original set on "Requirements" and "Rank".
Much of the effort dedicated to the testing was accomplished on the internal (development) test beds and on the Preview Test Bed, which also includes new components not yet ready to be deployed in production. In addition, a new approach to testing proved to be very effective. Scaling up to full production and real use cases from the experiments required, in fact, extensive troubleshooting and some additional development. A new concept, the Experimental Service, then emerged as a pragmatic testbed to be attached to the production infrastructure, by a selected number of users and immediately installed with all patches available from the developers. As of Easter 2007, the gLite WMS has proven to fulfill the performance and stability requirements expressed earlier this year by the acceptance criteria of the CMS and ATLAS experiments: a single WMS machine would need to sustain submission rates of at least 10K jobs/day (both with bulk and single job submission) with a failure rate less than 0.5% over a period of 5 days of continuous operation, with no manual interventions or restarts of the WMS services (and the L&B as well). During the 7 days when the tests were performed by CMS, some 115,000 jobs were submitted, yelding a throughput of 16,000 jobs/day (which is about 50% more than required) [Figure 2]. The limiter mechanism mentioned above kicked in a couple of times preventing further submission (but keeping the WMS system in a safe operation area). All but 320 jobs were properly carried out (0.3% failure rate). The resources were promptly reached, without appreciable delays and with a negligible fraction of failures due to the gLite WMS (most reported errors were application errors or site problems). On a later stress-test of the bulk submission the average throughput reached over one day was of 27,000 jobs.
Single job submission is also an important use-case for the submission of a limited number of jobs from a huge number of different users. It is also important to study how submission & MM times do actually scale in the present architecture. From the latest results, always performed by CMS, some 20,000 job/day have been submitted over a period of 11 days, reaching peaks of 22,000 jobs/day of throughput [Figure 3].

5. Current and future work

A significant part of the EGEE-II work has been devoted to portability, integration and deployment. In this respect, much effort was concentrated on the gLite 3.1 release and the completion of its transition to the ETICS integrated build and configuration system [16], started

in January 2007. The vast majority of the 32-bit gLite 3.1 release properly builds on ETICS and the 64-bit build issues are mostly being understood at the time of writing; also, there is ongoing work on the run-time testing of 64-bit nodes. This has allowed to address platform portability as well, i.e. the migration to Scientific Linux 4 (SL4). Platform portability was improved to the point that would make it easier to target any other kind of linux platfom at this point. Also, from the gcc run-time version in SL4, the WMS will particularly benefit from the optimized memory management for STL containers (memory/speed for list, red-black trees as used by sets and maps) which have a large effect when large number of resources populate the ISM. An alternate allocator [17], optimized for multi-threaded applications, has also been adopted to keep memory usage stable, hence not fragmented by the caching in "per-thread arenas", where deallocated memory is retained for later usage by the standard C allocator (ptmalloc2).
Support for some unused features (such as checkpointable and partitionable jobs) has been discontinued and some legacy components deprecated (like the Network Server, the legacy proprietary interface). A general restructuring of the code has started, triggered by the sharp decision to migrate to ETICS. This restructuring is not meant as an architectural change (all interfaces are preserved), but boils down to a desirable simplification of the stack that will simplify future work, including incorporation of additional functionality and porting to new platforms.
There is still ongoing work to further prevent and correct faulty situations for both services and jobs that might arise from increased activity. This involves improving reliability with a particular eye to implementing effective recovery mechanisms for all the supported job types.
No matter how high the throughput of a single WMS instance, higher orders of performance can always be reached, in a future scenario, by simply adding up new instances and referencing them in a round-robin fashion. In such a configuration, all the WMSes will be equally aware of the overall status of the Grid and their composition will cause neither interference nor fragmentation of the overall resource offer - something that can represent a scalabilty concern for other competing architectures. On the contrary, with the present model, performance will scale up almost linearly - the selection of an appropriate instance also being favored by the triggering of the load limiter which will make the User Interface switch to another WMS when needed (amongst a predefined given list or even using Service Discovery to proactively find active services), enabling in this way a further level of high-availability and load balancing.

6. Conclusions

The gLite WMS is designed and implemented to provide a dependable, robust and reliable service adopting open standards to promote interoperability among Grid services and allowing easier compliance with emerging protocols. The phase that ended with the acceptance test and the certification of the gLite WMS 3.1 provides a satisfactory platform, instrumented with a set of added-value features on top of Job Submission and with the flexibility of a Service Oriented Architecture. Development continues by integrating missing functionality (especially for the handling of collections), simplifying the code by reducing dependencies and keeping up with the standardization avnd definition of Grid services, procedures and protocols. We will also continue facing the challenge of reaching even higher levels of performance, scalability and reliability to find us prepared to meet the growing demand of the EGEE infrastructure.

References

[1] http://www.eu-egee.org/
[2] http://glite.web.cern.ch/glite/
[3] http://www.ws-i.org/
[4] http://www.ogf.org/
[5] M.J Litzkow, M. Livny and M.W. Mutka, *Condor-A hunter of idle workstations*, Proceedings of the 8th International Conf. On Distributed Computing, San Jose, CA USA (1988), pp. 104-111

[6] *JDL Attributes Specification*, https://edms.cern.ch/document/590869/1, EGEE-JRA1-TEC-590869-JDL-Attributes-v0-4
[7] http://www.cs.wisc.edu/condor/classad/
[8] www.globus.org
[9] J. Novotny, S. Tuecke, V. Welch, *An Online Credential Repository for the Grid: MyProxy*, IEEE International Symposiumon High Performance Distributed Computing; 2001 Aug 79; San Francisco, CA USA, pp. 104-114
[10] F. Dvorak, D. Kouril, A. Krenek, L. Matyska, M. Mulac, J. Pospisil, M. Ruda. Z. Salvet, J. Sitera, J. Skrabal, M. Vocu et. al., *Services for Tracking and Archival of Grid Job Information*, CGW05, Cracow - Poland, November 20 - 23, 2005
[11] DataGrid WP1 members (G. Avellino *et al.*), *The first deployment of workload management services on the EU DataGrid Testbed: feedback on design and implementation.*, Computing in High Energy and Nuclear Physics (CHEP'03), San Diego, March 24-28, 2003
[12] Rosario M. Piro, Andrea Guarise, Albert Werbrouck, An Economy-based Accounting Infrastructure for the DataGrid, Proceedings of the 4th International Workshop on Grid Computing (GRID2003), Phoenix, Arizona (USA), November 17th, 2003.
[13] V. Chiaschini et al., *An Integrated Framework for VO-oriented Authorization, Policy-based Management and Accounting*, Computing in High Energy and Nuclear Physics (CHEP'06), T.I.F.R. Mumbai, India, February 13-17, 2006.
[14] P. Andreetto, S. A. Borgia, A. Dorigo, A. Gianelle, M. Marzolla, M. Mordacchini, M. Sgaravatto, L. Zangrando et. al., *CREAM: a simple, Grid-accessible, job management system for local computational resources*, Computing in High Energy and Nuclear Physics (CHEP'06), T.I.F.R. Mumbai, India, February 13-17, 2006.
[15] http://grid.pd.infn.it/NA5/bes-wg.html
[16] M. E. Begin and et al., *Build, configuration, integration and testing tools for large software projects: Etics*, Springer Verlag Lecture Notes in Computer Science (LNCS) Series, LNCS 4401, pages 81-97, 2007.
[17] http://goog-perftools.sourceforge.net/

Figures

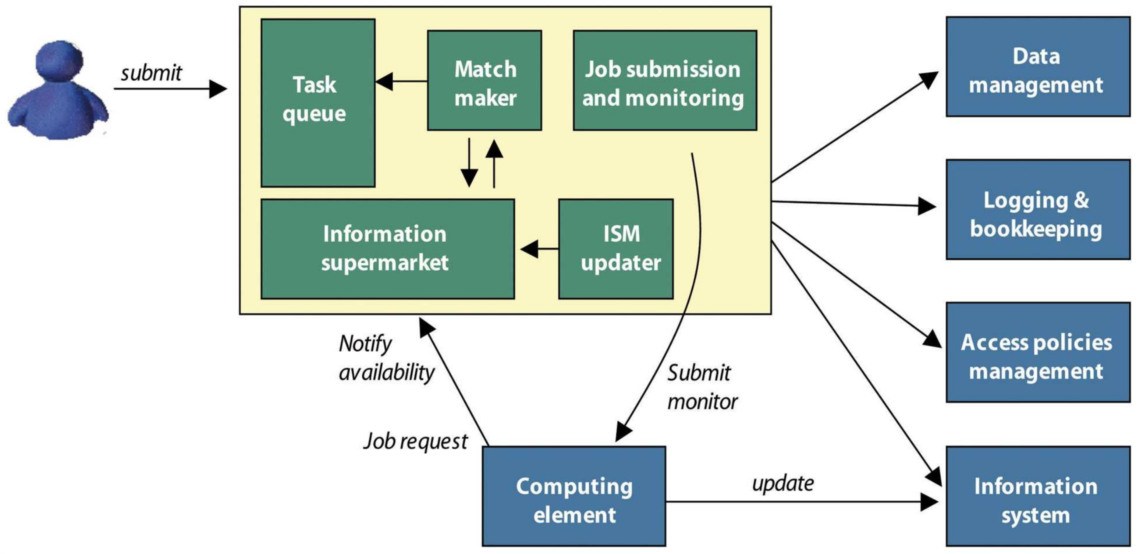

Figure 1. The gLite WMS internal architecture and its interactions with other Grid Services

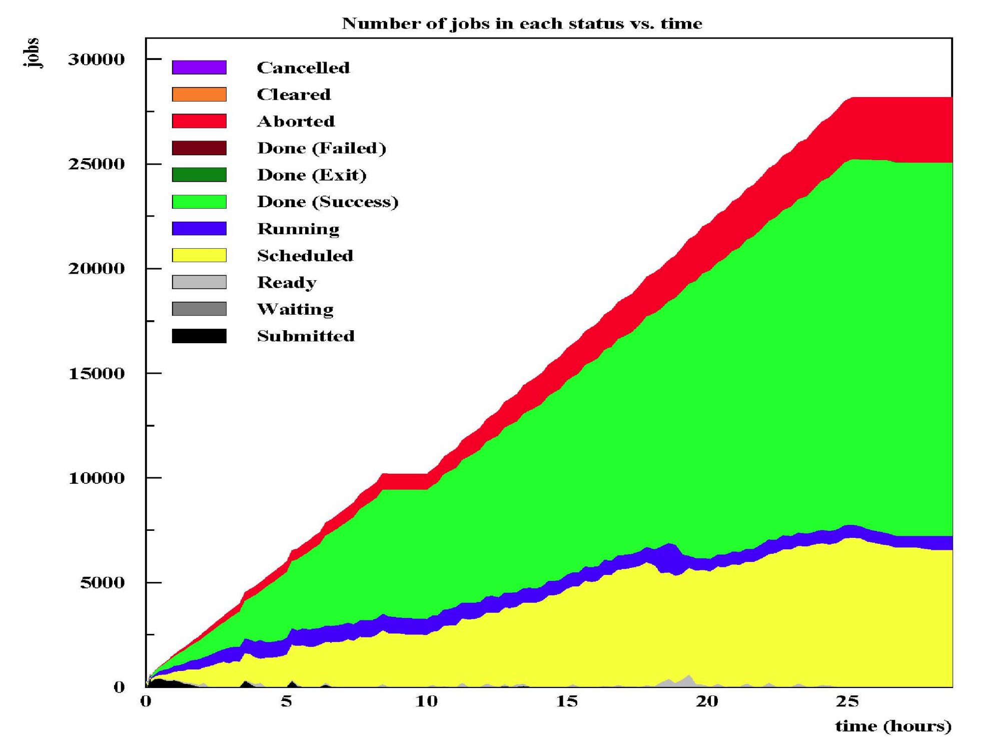

Figure 2. The number of processed jobs over one day of the week of the acceptance test

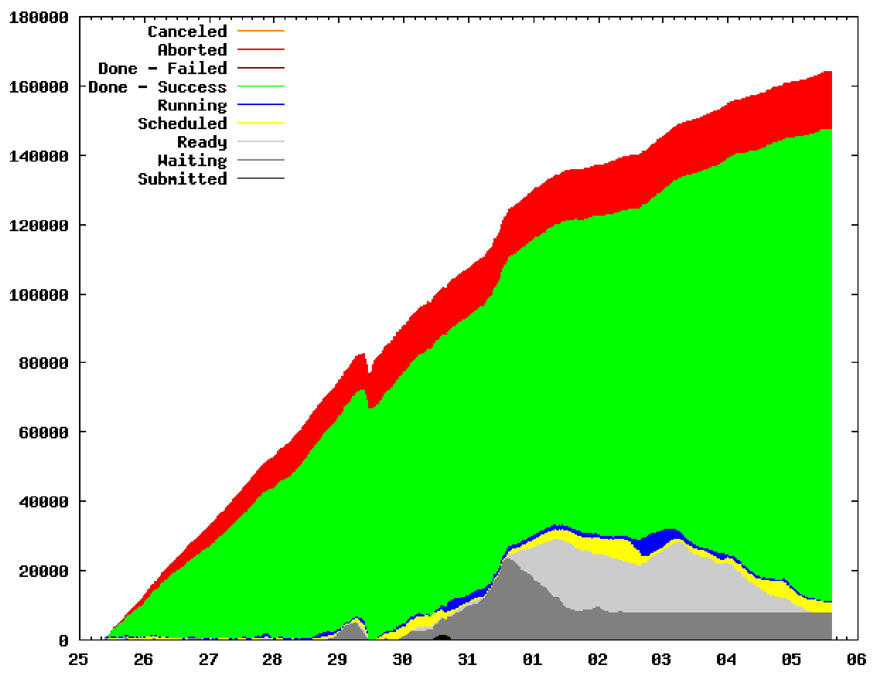

Figure 3. Processed jobs over day of the month, on a stress-test on single job submission by CMS