The gLite Workload Management System

Cecchi Marco[1], Capannini Fabio[1], Dorigo Alvise[1], Ghiselli Antonia[1], Giacomini Francesco[1], Maraschini Alessandro[2], Marzolla Moreno[1], Monforte Salvatore[1], Pacini Fabrizio[2], Petronzio Luca[2], and Prelz Francesco[1]

[1] I.N.F.N. - National Institute for Nuclear Physics - Viale Berti Pichat, 6/2 - Bologna (Italy)
[2] Elsag-Datamat s.p.a. - Via Laurentina, 760 - Rome (Italy)

Abstract. *The gLite Workload Management System represents a key entry point to high-end services available on a Grid. Being designed as part of the european Grid within the six years long EU-funded EGEE project, now at its third phase, the WMS is meant to provide reliable and efficient distribution and management of end-user requests. This service basically translates user requirements and preferences into specific operations and decisions - dictated by the general status of all other Grid services - while taking responsibility to bring requests to successful completion. The WMS has become a reference implementation of the "early binding" approach to meta-scheduling as a neat, Grid-aware solution, able to optimise resource access and to satisfy requests for computation together with data. Several added value features are provided for job submission, different job types are supported from simple batch to a variety of compounds. In this paper we outline what has been achieved to provide adequate workload and management components, suitable to be deployed in a production-quality Grid, while covering the design and development of the gLite WMS and focusing on the most recently achieved results.*

1 Introduction

Resource management and scheduling of distributed, data-driven applications in production Grid environments are challenging problems. The interested domains include workload management, resource discovery, brokering, accounting, authorization and authentication, 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 involving such huge scales and factors as the ones a production Grid has to deal with, has brought out non trivial issues requiring joint efforts with a strong degree of cooperation to be attained.
Grid computing technologies have been developed over the last decade to provide a computing infrastructure for a disparate and ever growing number of e-Science applications. A first large scale production Grid infrastructure was deployed by the Enabling Grids for E-SciencE (EGEE) [1] EU-funded project. Its operation was then further consolidated during its second phase (EGEE-II). The EGEE Grid infrastructure consists of a set of middleware services deployed on a

worldwide collection of computational resources, with an extensive programme of middleware re-engineering that has resulted in a consolidated software stack, gLite [2]. This long-standing project, now at its third phase (EGEE-III), will take further steps in moving Grids to dependable and sustainable production infrastructure while providing a continuous service to its expanding user base. EGEE-III will continue to develop gLite as its reference open-source middleware distribution.
In this paper we outline what has been achieved to provide adequate workload and management components, suitable to be deployed in a production-quality Grid, while covering the design and development of the gLite WMS, with particular respect to functionality and interoperability, focusing on the most recently achieved results.

2 The gLite WMS in a nutshell

The gLite WMS represents a key entry point to high-end services available on a Grid. It has been designed with some fundamental principles in mind: first of all aiming at providing a dependable and reliable service, where primary importance is given to 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 all those aspects that involve discovering the resources and selecting the most suitable ones, arranging for submission, monitoring and information gathering. In this respect, the WMS has to deal with a heterogeneous computing environment that in general encompasses different architectures and loss of centralized control, all this in presence of potential faults due to the distributed and diverse nature of the Grid environment, computers, networks and storage devices.

3 Functionality at various levels

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, which basically consist in execution requests, are usually referred to as "jobs". In a Grid environment the scope of such tasks/jobs needs to be extended to take into account other kinds of resources, such as storage or network capacity. The need for such a broader definition is basically due to the move from typical batch-like activity to applications with ever more demanding

requirements in areas like data access or interactivity, both with the user and with other tasks. In this respect, the WMS does support different types of jobs:

- Single batch jobs
- Work-flows: jobs with dependencies expressed as a direct acyclic graph (DAG)
- Collections: sets of jobs without dependencies grouped together and identified by a single handler
- MPI: based on message passing interface - a widely-used library to allow for parallel programming within a single cluster (intra-cluster)
- 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) [5]. The JDL is based on Classified Advertisements or *ClassAds* [6], developed within the Condor project [7], which basically consist 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 (physical and virtual memory, CPU, operating system, etc.). The user can then specify whatever attribute for the description of a request without incurring in formal errors, as ClassAds are not bound by any particular schema. Only a certain set of attributes are directly taken into account by the WMS on the base of documented semantics, the others will simply be passed on without specific processing. Also, the attributes used for describing high-end resources come from a common schema, the so called GLUE schema [8], born from a joint effort to standardize and facilitate interoperation between Grid infrastructures, e.g. the attribute "GlueCEStateFreeCPUs" will always indicate the number of free CPUs in all the resources making part of such a joint infrastructure.

Jobs are always associated with user proxy credentials and all 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 Virtual Data Toolkit from Condor and Globus [9] (VDT). While Condor plays a significant role in the present architecture as a job submission and tracking layer (see later), the Globus Security Infrastructure (GSI) is used throughout for enabling secure authentication and communication. GSI provides in fact libraries and tools for authentication and message protection that use standard X.509 public key certificates, public key infrastructure (PKI), the SSL/TLS protocol, and X.509 Proxy Certificates, an extension defined for GSI to meet the dynamic delegation requirements of Grid communities. A specific service, called Proxy Renewal and conceived as be part of the WMS, is devoted to renewing credentials, automatically and securely, for long-running jobs. This is a desired feature not to propagate throughout the Grid proxy certificates of a significant duration,

since they need to be reasonably longer than the expected duration of jobs, which in some cases can last for weeks, they are associated to. This scenario would obviously represent a security threat, but, on the other hand, working with short-lived certificates will cause long jobs to outlive the validity of their proxy and be consequentially aborted. To avoid this the WMS allows proxy certificates to be renewed automatically, when close to expiry, if the user allows the Proxy Renewal service to be enabled, this is done by specifying a MyProxy [10] server (the long-lived proxy keystore) in the job JDL. Another similar mechanism, implemented by the Job Submission Service (see later), is in place to forward freshly renewed certificates in the WMS instance to the Computing Element (CE, i.e. the Grid abstraction for a computing resource) where they will finally reach the Worker Node (WN, i.e. the machine where the job is actually executed).

The Grid is a complex system and things can go wrong at various stages of the so called submission chain. The WMS has been designed with the ability to recover from failures of the infrastructure by automatically resubmitting failed jobs, this is done at two levels. "Shallow" resubmission is utilized 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 network contact. "Deep" resubmission happens whenever a job fails after it started running; this situation 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 also have consumed a significant amount of (precious) CPU time. Users can therefore choose the number of times they will allow the job to be resubmitted in these two ways with two parameters of the JDL. If a job fails after having reached the maximum number of retries it will be terminally aborted.
Submitting a job actually means passing its responsibility to the WMS whose purpose is then finding the appropriate resource(s) matching user 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 the selected set of CEs for execution. The decision about which resource is adequate to run the job is the outcome of a so called match-making process between the "demand", represented by the submission requirements and preferences, and the "offer", represented by the characteristics of the available resources. The availability of resources for a particular task depends not only on the actual state of the resources, but also on the utilization policies that the resource administrators and/or the administrator of the Virtual Organization (VO) the user belongs to have defined for each of their users. It can happen, not rarely, that none of the resources available on the Grid at a given time is able to satisfy some job's requirements, in suchcase the submission request is kept pending by the WMS and periodically retried, the retry period being a configuration parameter, until the request expires.

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 so that if a user (which can, by the way, also be represented by an automatic system) has no matching resources it can temporarily stop submitting. Request status follow-up can be achieved through the Logging&Bookeeping service (L&B) [11], another key service responsible for tracking jobs in terms of events (important points of job life, e.g. submission, transfer from a WMS component to another one, finding a matching CE, starting execution etc.) gathered from various WMS components as well as other Grid services. Each event type carries its specific attributes. The entire architecture is specialized for this purpose and is job-centric: any event is assigned to a unique Grid job identifier. The events are gathered from various WMS components by the L&B producer library, and passed on to the locallogger daemon, running physically close to avoid any sort of network problems in a store&forward fashion.

All the various job management tasks mentioned so far are accomplished by different components basically implemented (mostly in C++, with extensive usage of the Boost [12] libraries) as different processes or threads, all communicating via persistent data structures [Figure 2]. As anticipated, one core component is the Match-Maker which sorts out a list of resources satisfying the given requirements. These resources might even include Storage Elements (SE, i.e. the Grid abstraction for a storage resource) if the user requested to need manipulate data. Such returned list of suitable resources is ordered, given that more than one resource could match the specified requirements. The highest-ranked resource will typically be used. The ranking function is provided by the user in the JDL. Just a trivial example how a ranking expression would look like in the JDL:

$$Rank = -other.GlueCEEstimatedResponseTime; \tag{1}$$

will indicate to send the job to the resource with the lowest estimated queue traversal time.
To avoid the top-ranked resource to be repeatedly chosen upon successively close in time requests, so becoming overrated, before the Local Batch System and the Information Systems could in turn update such dynamic information, a stochastic algorithm can be used to perform a smoothed selection among all the matching resources - weighted according to their actual rank - in such a way to prevent congestion for the initially best ranked resources.

Proper handling of massive data volumes is a very important aspect in production quality Grids (it is maybe worth noting that one of the projects from which EGEE originates was called "DataGrid"). The JDL allows the definition of requirements based on data through an attribute called "DataRequirements" which is structured in such a way to allow users to target experiment-specific catalogs for their jobs and to mix different input data types supported by different data catalogs in the same job description. Logical File Names (LFN), Grid Unique ID-entifiers (GUID), Logical Dataset (LDS) and/or generic queries can

be used to retrieve data from SEs. All of them are used by the WMS to query the related Data Catalog for getting back a list of Physical File names (PFN) that are needed by the job as input for processing ([13] for more information). Output data can then be stored to a specified SE and registered to a catalog. While match-making is made between two entities - typically the job and the computing resource, another interesting feature relating data management, called gang-matching allows to take into account, besides CE information, also SEs in the process. A typical use case for gangmatching might be: a job has to run on a CE close to a SE with at least 300 Mb of available space. This translates into a JDL statement like the following:

$$Requirements = anyMatch(other.storage.CloseSEs, target.GlueSAStateAvailableSpace > 300); \quad (2)$$

Getting closer to the core business, one of the most important tasks performed by the WMS is, needless to say, scheduling (some would prefer call it planning, or meta-scheduling). More or less "eager" or "lazy" policies can be supported in this respect. 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 (hence requiring asyncronous communication with the Information Provider), 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". These two approaches are quite symmetrical indeed: eager scheduling implies matching a job against multiple resources, whereas lazy scheduling implies matching a resource against multiple jobs.
The WMS is potentially able, by design, to work with each of these two opposite modes. They both represent a neat grid-aware solution for job scheduling even if, in the course of time, the 'push-mode' emerged as the one and only method actually utilised in the production infrastructure (maybe due to the fact that pull-mode requires asynchronous Information Providers and that some care would be needed to handle notifications to more than just one WMS instance to allow for scalability and to prevent working with a single point of failure). For the record, other Grid meta-scheduling systems are able to enable late binding, apparently much like the pull-mode would behave. Actually such systems, sometimes referred to as "pilot-jobs" frameworks, implement sort of shortcut where a single VO-level scheduler submits "neutral" placeholder jobs - so keeping a constant pressure onto all the available resources - which, once running on the WN, are able to finally call forth end-user jobs. Of course such pilot jobs are seen (and accounted) by the Grid infrastructure as any other user job. A thorough analysis of the *pro et contra* of such emerging scheduling models would be out of the scope of this paper, nevertheless, other than being affected by security implications, they cannot really be considered as an alternative to the pull-mode, in any case, being just a custom layer built on top of the very same infrastructure. Apart from serious security implications which will not be addressed here, one way or

the other pilots need to be scheduled within the Grid services and protocols, i.e. a Grid meta-scheduler (direct job submission cannot be considered at this level a Grid-aware solution).
Back to the WMS, the mechanism that allows for a flexible application of such different policies as the push or the pull mode is the decoupling between the collection of information about resources and its usage. This is enabled by a repository of cached resource information collected from the various supported Information Providers, called Information Super-market (ISM), which is available in read-only mode to the match-making engine and whose update can be the result of either the arrival of notifications or active polling on resources or some arbitrary combination of both from different source of Information Providers. The ISM represents one notable improvement in the WMS as inherited from the EDG and LCG projects where the information was collected in real-time - so contacting Information Providers for each single request, in a less efficient and reliable fashion.

Reflecting the demand-offer/job-resource symmetry, each single job request is kept in a event based priority queue (different request types have in fact different priority), which recently replaced a data structure called task-queue (TQ, inherited from Alien [14]). This allowed us to remove several locks throughout, once needed to keep the TQ synchronised, and now requests (coded as functors to be executed by a thread pool) line up as soon as they arrive waiting to be processed as stateless as possible, according to the specific situation and/or error condition, while preserving the ability to hold a submission request if no matching resources are immediately found. Such periodic activities (timed events) will in fact re-schedule themselves to show-up at a programmed later time in the priority queue.

Another interesting feature, which has been added quite recently, is represented by the so called "bulk match-making". This optimisation, enabled for collections, allows to perform the match-making for each subset of jobs sharing same characteristics instead of matching each single job. The original collection is partitioned into such subsets according to some significant attributes (JDL attribute "SignificantAttributes") which will identify by the equivalence classes. A typical use-case for specifying significant attributes could be, as an example, parting the original set on "Requirements", "DataRequirements" and "Rank".

Here is a summary of the more relevant functionalities implemented in the gLite WMS:

- Resubmission: shallow or deep
- Stochastic ranking
- Bulk-submission and bulk match-making
- Proxy renewal
- 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 based on machine's vital parameters
- Automatic sandbox files archiving/compression and sharing between jobs
- Match-making with data
- Gang-matching

4 Interoperability and interfacing

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 all the involved entities. 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, which defines a set of Web Services specifications to promote interoperability. This newly introduced Web Service based implementation replaced a legacy network interface based on a proprietary protocol.It manages user authentication/authorization and operation requests. It runs in an Apache [15] container extended with FastCGI [16] and Grid Site [17] modules. The Fast CGI module implements Common Gateway Interface (CGI) functionality along with some other specific features. The most important advantages of using FastCGI are its performance and persistence. FastCGI applications, in fact, 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) [18] and file transfer over secure HTTP. It also provides a library for handling Grid Access Control Lists (GACL). The Web Service hosting framework provided by Apache, Grid Site and gSOAP has allowed the development of this front-end interoperable service in C++, giving continuity and consistency with the rest of the coding.

About interoperation with other Grid services, we need to describe in more detail how job management is accomplished by the WMS. A service called Job Submission Service (JSS) is responsible to actually establish an authenticated communication with the selected resource to forward the job and to monitor

its execution. To implement such lower level layer Condor-G has been always adopted. A monitoring service, part of the JSS, is also responsible for watching the Condor log files intercepting interesting events concerning active jobs which affect the job state machine and trigger appropriate actions. Every CE supported by Condor-G is then implicitly supported by the WMS as well, in particular the LCG CE (pre-Web-Service Condor-G plus GRAM on the CE) and the gLite CE (pre-WS Condor-G plus Condor-C on the CE). Recently, with the advent of the newest WS-I/BES [19] CE called CREAM [20], a new component of the WMS suite, called Inteface to CREAM Environment (ICE), has been introduced as part of JSS for job management towards CREAM. ICE is a gSOAP/C++ layer which will securely manage job operations to CREAM CEs. In doing so, it subscribes to the gLite CEMon information system [21] in order to asynchronously receive notifications about job status changes. ICE also performs synchronous status polling for unresponsive jobs, in case some notifications are lost.
Interoperation with Information Providers is achieved either syncronously or asyncronously for those providers who support it. We actually do provide interfacing with the Berkely Database Information Index (BDII), support for other providers has been recently dismissed due to lack of use.

About formalisms for defining jobs, the WMS fully endorses the Job Submission Description Language (JSDL). This is an emerging OGF standard which aims at facilitating interoperability in heterogeneous environments, through the use of an XML based job description language that is free of platform and language bindings. JSDL contains a vocabulary and normative XML Schema that facilitate the expression of job requirements and preferences as a set of XML items. What happened in the past and still can happen is that several different organizations accommodate a variety of job management systems, where each system has its own language for describing job submission. This represents a severe obstacle for interoperability. In order to utilize such different systems altogether the involved organizations would have to prepare and maintain a number of different job submission documents, one for each system, basically all describing the same operations. The JSDL represent a significant effort toward unification 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.

On the front of Grid interoperability, having already set up a long-standing interaction with OSG, recent work has been done to enable interoperability with both NorduGrid [22], and its ARC CE, and UNICORE [23], with a contribution to the writing of the Grid Interoperation Now (GIN) [24] profile. More pragmatically, much of the issues concerning interoperability reflects in the way the WMS job-wrapper (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.) is engineered. Due to the diverse nature of resources belonging to one or more

Grids, such script must be kept as simple and as robust as possible. The job-wrapper may in fact be running in an unfriendly WN 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 series of hooks is provided in the jobwrapper generation procedure, allowing specific customisations to be inserted by users, VO managers and site administrators. This approach reduces hard-coding, by decoupling general and specific operations, without limiting functionality. For users, prologue and epilogue scripts have been included - to be run before and after the job is executed - basically with the intent of setting and cleaning up the proper environment for "real" jobs; for VOs, a customisation point is foreseen mostly used to hook up the proper middleware version; for similar purposes resource managers are allowed to hook up their scripts throughout several strategic points of the job-wrapper.

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

- Backwards compatibility with LCG-2
- Automatic renewal of credentials
- GridFTP and HTTPS to handle secure file transfer for the sandbox
- Service Discovery for obtaining new serivice 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
- Support for Grid Site delegation 2.0
- Interoperability with the american Open Science Grid (OSG), Nordugrid and UNICORE
- Integration with Grid accounting and autorization frameworks
- User prologue/epilogue scripts accompanying the job, more custom scripts allowed to be hooked for use by resource and VO administrators

5 Results and future developments

As of late 2008, the WMS has been deployed in a large number of multi-user and multi-VO scenarios. Thorough testing and intense troubleshooting have been accomplished during all these years, of course driven by the compelling needs of the LHC experiments. This has led to a significant level of service stability for the current production release. Much of this effort was accomplished using the development test-bed and the preview test-bed, which also includes new components not yet ready to be deployed in production, as it was the case for ICE.

In addition, the concept of Experimental Service proved to be very effective: a development instance, attached to the production infrastructure, to be accessed by a selected number of users and immediately installed with the latest available patches.

Now that an acceptable level of sustained stability has been reached, work is being done to further target performance. In particular, after the (effective) introduction of collections, the average match-making time, performed on the full production BDII, has room to improve, especially for single jobs. The next to come release will be able to perform the match-making in parallel thanks to a re-design of the ISM that will be doubled in order to remove some locks with a huge scope at the moment necessary to keep the structure synchronised with readers and writers insisting on it. A read-only copy will be available for readers, the request handlers needing to perform the match-making, while another one will be created in background while purchasing. A pseudo-atomic swap between these two copies will occur periodically and timedly so that the ISM at the moment accessed by reader threads is disabled while, in the mean-time, the freshly purchased one, since then only accessed for writing, will then become available to the readers only. Two ISM instances will be contemporarily present in memory only for limited period - the time needed to carry out purchasing and to wait for the older threads, still pointing to that very copy, to complete - after which such instance can be definitely cleared. Such a design has already been stress tested in a prototypal instance installed as an experimental service, with the collaboration of the CMS experiment; a peak performance of about 100.000 jobs/day were reached for more than two consecutive days [Figure 3].

Also, one of the plus points of pilot-based job scheduling is the ability to match jobs to resources very quickly, as compared to our present gLite WMS. This can basically be done by virtue of the fact that the VO decides user prioritization in advance in such a way that as soon as a pilot on a resource signals its availabiliy to get new jobs, the VO scheduler just scans the job requests list, which is ordered according to a VO-wide policy, so that it can simply stop to the first job matching the requirements. Nothing prevents the gLite WMS to act in a similar way; in fact, the WMS allows each single user to specify a rank function to apply to his jobs, as we have already seen. This is a nice feature, nevertheless it requires matching against the whole ISM, not simply stopping to the first one. Provided a fixed rank (i. e. VO-based, much like pilot jobs frameworks work), the WMS could instead keep the ISM indexed accordingly so that the match-making could just stop to the first matching resource, which at that point will be also the highest ranked one. This will dramatically reduce the time for match-making (stochastic ranking could be done in any case truncating at the first n matching resources). This new model will represent a further step toward providing added value in such a way that, in a future scenario, the WMS will be even able to automatically find the best effective rank for its jobs, using some feed-back mechanism to rate resources according to their performance measured

over the entire job's life-cycle (i.e. there is no feed-back at the moment about the quality of status information as published by the Information Provider).

Nonetheless, thanks to the modularity of its design, the present WMS architecture allows for scalability in a more flexible way than pilot submission frameworks. The VO pilot scheduler, in fact, other than being a single point of failure, needs to be able to manage the resource domain space in its entirety. Two different pilot schedulers would require parting the resource domain space to work together, with the consequence of fragmenting the computing offer into two separate sets. On the contrary, several gLite WMS instances can work together over the whole production infrastructure, the total throughput scaling up in an almost linear fashion. Stochastic ranking could be eventually utilised to minimise latencies coming from the Information System update rate. In fact, this can be done, as compared to the pilot-based approach, right because each WMS instance would get status information by interoperating with a specific Grid service (the Information System, as said) and not directly from pilot jobs.

The next gLite WMS release (3.2), under preparation at the time of writing, will contain several improvements, as the result of the intense restructuring activity which took place during EGEE-II, not yet fully ported into the release branches. Among other things, this new release, aimed at providing a more lightweight and responsive service thanks to a significant redesign of its core component, will be instrumented with all the aforementioned parallel matchmaking, IPv6 compliancy, support for Grid Site delegation 2.0 and it will be Scientific Linux 5 ready. An official Web Site [25] and Twiki pages have been set up, being kept readily updated, for documentation and support about all the activity concerning the WMS.

6 Conclusions

The gLite WMS is designed and implemented to provide a dependable, robust and reliable service for efficient distribution and management of end-user requests for computation, storage, network, instruments and whatever resource may be shared across a production quality Grid. It comes with a fully-fledged set of added-value features to enhance low-level job submission. Thanks to the flexibility of a scalable, fault-tolerant and service-oriented architecture it has been deployed in a number of layouts and scenarios.
After seveal years of operation the WMS has reached sustained stability and a performance targeted at covering the current needs, coming in particular way from High Energy Physics and Bioinformatics. Development continues by supporting enhancements requests expressed by the increasing number of experiments and users of the EGEE community, keeping up with the standardization and definition of Grid services, compliancy to emerging formal and *de-facto* standards 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. *JDL Attributes Specification*, https://edms.cern.ch/document/590869/1, EGEE-JRA1-TEC-590869-JDL-Attributes-v0-4
6. http://www.cs.wisc.edu/condor/classad/
7. 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) 104-111
8. *http://forge.gridforum.org/sf/projects/glue-wg*
9. www.globus.org
10. J. Novotny, S. Tuecke, V. Welch, *An Online Credential Repository for the Grid: MyProxy*, Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE, 2001
11. 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
12. *www.boost.org*
13. https://edms.cern.ch/document/487871
14. S. Bagnasco, P. Cerello, R. Barbera , P. Buncic, F. Carminati, P. Saiz, *AliEn - EDG interoperability in ALICE*, CHEP-2003-TUCP005, Jun 2003, 3pp
15. *http://www.apache.org*
16. http://www.fastcgi.com
17. *http://www.gridsite.org*
18. *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*
19. *http://grid.pd.infn.it/NA5/bes-wg.html*
20. *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.*
21. *CEMon, http://grid.pd.infn.it/cemon/field.php*
22. http://www.nordugrid.org/
23. *http://www.unicore.eu/*
24. http://forge.ogf.org/sf/projects/gin
25. *http://web.infn.it/gLiteWMS/*

Figures

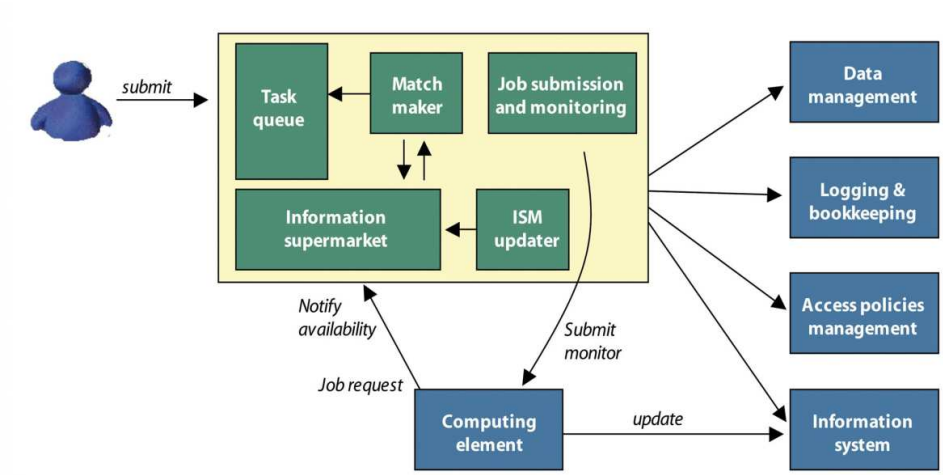

Fig. 1. A schetch of the gLite WMS internal architecture showing its interactions with other Grid Services

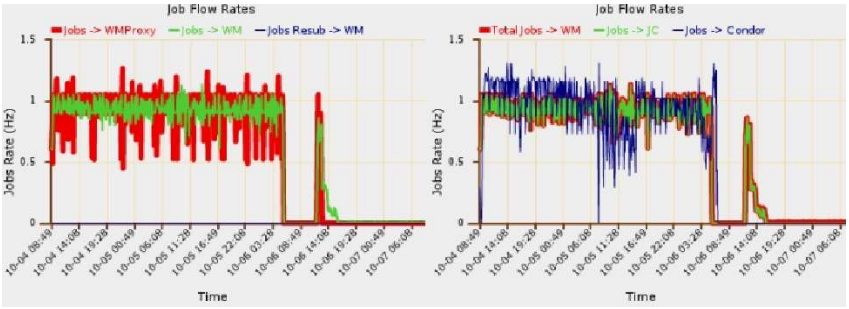

Fig. 2. Throughput of about 90.000 jobs/day (>1 Hertz rate as shown by the plot) over a period of more than two days on a stress test by CMS