FLEXIBLE JOB SUBMISSION USING WEB SERVICES: THE GLITE WMPROXY EXPERIENCE

G. Avellino, S. Beco, A. Cavallini, A. Maraschini, F. Pacini, A. Parrini, C. Scarcella, M. Sottilaro, A. Terracina, DATAMAT S.p.A.
F. Dvorak, D. Kouril, A. Krenek, l. Matyska, M. Mulac, J. Pospisil, M. Ruda, Z. Salvet, J. Sitera, J. Skrabal, M. Vocu, CESNET z.s.p.o.
S. Monforte, M. Pappalardo, INFN Catania
S. Andreozzi, M. Cecchi, V. Ciaschini, T. Ferrari, F. Giacomini, R. Lops, E. Ronchieri, V. Venturi, INFN Cnaf
G. Fiorentino, V. Martellli, M. Mezzadri, E. Molinari, F. Prelz, D. Rebatto, INFN Milano
P. Andreetto, A. S. Borgia, A. Dorigo, A. Giannelle, M. Marzolla, M. Mordacchini, M. Sgaravatto, L. Zangrando, INFN Padova
A. Guarise, G. Patania, R. Piro, A. Werbrouck, INFN Torino

Abstract

Contemporary Grids are characterized by a middleware that provides the necessary virtualization of computation and data resources for the shared working environment of the Grid. In a large-scale view, different middleware technologies and implementations have to coexist. The SOA approach provides the needed architectural backbone for interoperable environments, where different providers can offer their solutions without restricting users to just one specific implementation. The WMProxy (Workload Manager Proxy) is a new service providing access to the gLite Workload Management System (WMS) functionality through a simple Web Services-based interface. The WMProxy was designed to efficiently handle a large number of requests for job submission and control to the WMS and the service interface addresses the Web Services and SOA architecture standards, in particular adhering to the WS-Interoperability basic profile. In this paper we describe the WMProxy service: from its architecture, independent from the used Web Services container, up to the provided functionality, all together with the rationale behind the decisions made during both the design and implementation phases. In particular, we provide a description of how the WMProxy is integrated with the gLite Workload Management System; the used technologies, focusing on the Web Services features; the mechanisms adopted to improve performances still keeping high reliability and fault-tolerance; the changes in the job submission operation chain with respect to the previous generation of Workload Management Systems and the new operations provided in order to support bulk-submission and improve Client-Server interaction capabilities.

INTRODUCTION

In the context of the EU funded EGEE project, in order to address the need of a distributed scheduler and resource manager for a Grid infrastructure, a Workload Management System, the gLite WMS, was designed, implemented and deployed also by integrating existing technologies [1].

The aim of WMProxy is to provide a new component to access the gLite WMS. The need of such a component, besides adhering to the emerging design methodology called Service Oriented Architecture (SOA), is to improve the performance of existing similar components, to handle more efficiently a large number of requests for job submission and control and also to provide additional features.

The WMProxy is implemented as a Web service. The Web is considered a successful technology mostly because of its simplicity and ubiquity. A Web service allows us to take advantage of the benefits of the Web, not only to provide information, but also to offer services to a greater community of possible users.

Since the EGEE Middleware is a composition of Grid services provided by different vendors and/or operated by different organizations, the Web services technology is a powerful means to achieve service interoperability and allow easier compliance with emerging standards such as OGSA and WSRF.

WMPROXY ARCHITECTURE

To adhere to the SOA model, WMProxy has been designed and implemented as a SOAP Web service. The interface is described through the Web Service Description Language (WSDL). The WSDL file was written following the Web Services Interoperability Basic Profile (WS-I Basic Profile). This profile defines a set of Web Services specifications that promote interoperability [2].

The WMProxy service runs in an Apache container extended with FastCGI and GridSite modules.

The FastCGI module [3] provides Common Gateway Interface (CGI) functionality with some other specific features. The most important advantage of FastCGI is its

performance and persistence. FastCGI can be seen as a new implementation of CGI, designed to overcome CGI's performance problems. The major implementation differences are:

- FastCGI processes are persistent: FastCGI applications are able to serve multiple requests
- Application instances are spawned/killed dynamically according to demand
- FastCGI protocol multiplexes the environment information, standard input, output, and error over a single full-duplex connection.

WMProxy in its default configuration is run as a dynamic FastCGI application in order to take full advantage of its capacity to accommodate request loads. However, it can be run in any of the available FastCGI modalities.

WMProxy is implemented as a single-threaded application thus ensuring more reliability and robustness than multi-threaded applications; thanks to the FastCGI process manager the web server maintains a pool of processes that allow improving performances as generally done through multi-threaded application.

GridSite [4] provides a module extending the Apache webserver for use within Grid frameworks by adding support for Grid security credentials such as GSI and VOMS, and file transfer over HTTPS. It also provides a library for handling Grid Access Control Lists (GACL).

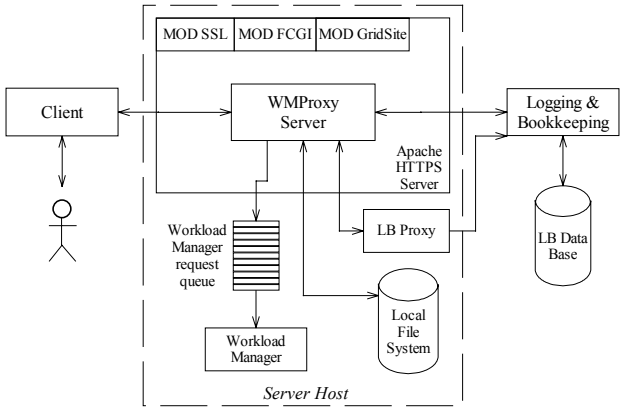

Figure 1: WMProxy Integration with the WMS.

The Web Service hosting framework provided by Apache, GridSite and gSOAP [5] has allowed the development of the WMProxy service using the C++ language. The choice of the implementation language has been driven mostly by the need to re-use and integrate existing production-quality components and libraries. A further boost in this direction has been given by preliminary tests that have shown better performance for the couple C++/gSOAP in comparison with the most commonly used framework for the development of Web Services (e.g., Axis/Java). Last but not least the gSOAP interoperability with a variety of other SOAP implementations and toolkits that permitted the development of a service accessible from clients written in different programming language and running on any computer platform.

The integration of the WMProxy within the WMS is shown in Figure 1. WMProxy provides a core module performing validation, conversion, environment preparation and information logging for each incoming request, before delivering it to the Workload Manager (WM). WM is the core component of the WMS taking the appropriate actions to satisfy incoming requests. Communication between the WMProxy and the WM occurs through a thread-safe, file-system based queue. The LBProxy, which is used as a state storage of active jobs, is the only external service with which the WMProxy interacts. The LBProxy service provides an optimized access to the Logging and Bookkeeping Service (LB). Other relevant information about processed jobs/requests are stored in the local file system in a reserved area managed by the WMS.

WMProxy is composed of different modules that are described here below. The *gSOAP Layer* module implements the gSOAP calls to the provided operations, and it is the only WMProxy module that is gSOAP dependant. This module can be seen as an intermediate layer between gSOAP and the WMProxy core implementation.

WMProxy Operations is the core module of the service providing the actual implementation of the service operations. It uses all the remaining modules to perform all actions needed to serve an incoming request.

The *Authorization* module performs the authorization control of incoming requests. This is done taking into account the user's DN and FQAN read from the proxy certificate presented by the client and checking for the corresponding entries in the service GACL files. The Authorization module also performs the calls to the LCMAPS library when Grid-to-local user mapping is needed.

The Directory Manager is the module used for creating and managing the job reserved area on the file system. Since all the operations on the file system related to a job have to be performed as the local users mapped to the grid user owning the job, the Directory Manager is run through a sort of sudo callout by the WMProxy that instead runs normally as an unprivileged user.

The *LB Access* is the module responsible for the interaction with LB and LBProxy. This interaction occurs for job registration, logging of job-related events and to get information about the job life-cycle (i.e., to query for the job status or previously logged events).

The *Request Delivery* module is used to write processed requests into the WM requests queue. This module is able to handle plug-ins to provide delivery to possibly different WM and/or requests queue implementations. The *Delegation* module implements the operations exposed in the imported delegation WSDL description using primitives provided by GridSite. It is

also used to manage the cache area for delegated credentials held by the WMProxy service.

ACCESSING WMPROXY SERVICES

WMProxy provides a set of core operations relevant to job submission and job control. It also makes available a large set of auxiliary operations. They are all described and documented in the service WSDL [6].

The authentication and authorization processes, that are common to all WMProxy operations, and the delegation process, which is requested for the jobListMatch, jobSubmit and jobRegister operations, are briefly described in the following two sub-sections. The rest of the paper focuses instead on how these and other operations are composed to obtain the job submission chain.

Connection Security, Authentication and Authorization

Message exchange between client and server is performed with SOAP/HTTPS.

Authentication of the requesting user is done through the X.509 proxy certificate signed by a trusted Certification Authority (CA), which guarantees that the exposed public key is really owned by the user.

The authentication process is handled by the Apache HTTP server by means of the SSL and GridSite modules. The authenticated request together with information about the checked credential (e.g. expiration time, VOMS extensions), are then passed to WMProxy within the CGI environment. WMProxy does not need to directly manipulate Grid credentials in this phase.

Authorization is implemented through the GACL library, which is provided by GridSite for manipulating Access Control List (ACL) files. Authorization can be either FQAN (coarse-grained) or DN-based (fine-grained) according to the type of proxy presented by the client. There are two authorization steps that are performed for an incoming request within the WMProxy. The first one assesses whether the requesting user is authorized to use the WMS services. The second step, which is only performed for operations directly related to a given job, assesses whether the requesting user is either the owner of the job or has permissions to manage it.

Proxy Credential Delegation

The delegation is the process used to transfer rights and privileges to another party. Since the WMProxy and the WMS when providing some services need to interact with other services, operating on behalf of the user, a delegation process is needed to transfer client proxy credentials to the server host. The delegation service is provided through a port type whose description is imported into the WMProxy WSDL file from the gLite common delegation WSDL file.

Delegated credentials are uniquely identified by the association of the delegation identifier, provided by user, and the user's DN within the credentials.

Multiple delegations of the same proxy credential are allowed with different delegation identifiers; however, it is recommended to do it once at the beginning of the working session and reuse the same delegation identifier, as delegation process is generally time-consuming.

The WMProxy holds a cache of the delegated proxies, which is purged periodically from the expired credentials; upon a submission request the service performs a mapping between the incoming job and a proxy in its cache according to the requesting user DN and the specified delegation identifier. From that point on, each operation performed for that job is done using the credential associated to it in this way.

Job Submission

In this section we describe the sequence of WMProxy operations and the corresponding actions that need to be performed to accomplish job submission to the WMS, i.e. the delivery of the user's job request in the WM request queue.

Job submission requires both, a description of the job to be executed, and a description of the needed resources. These descriptions are provided with a high-level language called Job Description Language (JDL) [7].

The actual submission is done by calling in sequence the two service operations *jobRegister* and *jobStart*

When a *jobRegister* request arrives to the WMProxy, if the client has the rights to proceed, a job identifier is generated and a set of specific attributes needed by the WMS for handling the request appropriately are inserted in the job description. The requesting user is then mapped to a local user by means of LCMAP so that the job local directories and files can be created with appropriate ownership and permissions. When all the aforementioned steps have been successfully completed, the job with the generated job identifier and the enriched JDL description is registered to the LB and from that point on is uniquely identified and can be followed-up throughout the whole system with its identifier. Afterwards, the job is registered for proxy renewal (if requested) and the job handle is returned to the user. In case the JDL description provided by the user represents a compound request (see next section), i.e., a set of jobs, a tree of identifiers is returned. This tree contains the job identifier of the single jobs composing the group, and a job identifier to identify and manage the entire group.

The completion of the WMProxy job registration phase is the sign that the job has been taken into account within the Grid environment and is ready to be actually submitted to Grid resources. This further step is triggered by the invocation of the *jobStart* operation. During the start operation the provided job identifier is used to check if the corresponding job has the pre-requisites for being started, i.e., it has been previously registered by the same WMProxy service and it has not already been started. Only if both conditions are met or previous job start attempts have failed for some reason the job processing is carried on by further manipulating the job description, decompressing, extracting, locating sandbox files in the

appropriate areas and lastly writing the request into the WM requests queue.

Compound jobs start requires additional steps to be performed before passing the request to WM. These are the registration to LB of all sub-jobs composing the request and the creation of the job reserved area on the file system for all of them.

All auxiliary actions needed for the job to run successfully on the remote resource can be performed in the phase between job registration and job start. Relevant examples could be the preparation of the input files needed by the job at run time including both the job input sandbox files and the data stored on Storage Elements (SE) and registered onto Grid Catalogs.

In particular the job input sandbox files can be either located on accessible external file server or have to 'travel' together with the job from the client machine to the resource where the job is going to run. In the latter case the needed files have to be uploaded by the user to the WMS node so that they can be then managed by the WMS and made available to the job. WMProxy provides the URI of the files destination location through specific operations. Supported file transfer protocols are gsiFTP and HTTPS.

The split of the job submission process into the two well distinct phases, job registration and job start, realizes a sort of two-phase commit: with the former operation the job enters the system remaining registered for a configurable period of time; with the latter one, the job can utilize the system. This gives the user full control over single submission phases; in case of failure single operations can be performed again separately. Moreover it allows performing preparatory activities when the job has already entered the system and moving specific time-consuming processing after the operation response has been provided to the client.

IMPROVEMENTS

In order to address the growing requests for job submission performance improvements and efficient management of very large number of jobs/requests, a series of features have been introduced within WMProxy.

Most notable are related with:

- "Bulk" job submission
- Job's sandboxes management
- Asynchronous job start operation

They are briefly described in the following sub-sections.

Bulk Submission

Bulk jobs submission within WMProxy is based on the newly defined requests types, job *Collections* and *Parametric* jobs, and relies on the support for *Direct Acyclic Graph*s (DAG) provided by the WMS.

A DAG is a is a set of jobs where the input, output, or execution of one or more jobs depends on one or more other jobs. The jobs are graph nodes (vertices), the edges (arcs) identify the dependencies. A collection is a group of jobs without dependencies. The grouping concept is here fully user-defined. Lastly a parametric job is a job having one or more attributes in its description that vary their values according to a set of parameters. A parametric job represents a set of very similar jobs only differing for the values assumed by some JDL attributes.

The JDL and the library for manipulating it have been subjected to significant changes and evolutions to handle the aforementioned new request types.

Upon submission of one of such requests, which is done by means of a single description, the WMProxy service uses the JDL library to validate the request and convert it into a DAG (possibly without dependencies) of jobs that can be handled by the WMS.

This approach enables the submission of a large number of jobs through a single submission, saves from the overhead of multiple connections to the service (i.e., authentication and authorization), allows sharing of sandboxes files between jobs (see next section) thus reducing the size of data to be transferred and makes available a single handle to follow-up and control the whole set of jobs generated by the served request.

Job's Sandboxes

This feature is particularly useful when submitting compound requests (i.e., DAGs, collections, parametric jobs). The improved JDL library allows indeed the WMProxy to recognize those files that are common to the input sandboxes of different sub-jobs of the requests, to locate them appropriately and rearrange the jobs descriptions so that the files are accessible at run time to all jobs needing them. This, together with the WMProxy capability to deal with jobs (both single and compound) whose sandbox files are uploaded on the WMS node as compressed archives of files allows minimizing the number of calls to the available file transfer services and lowering the size of data to be transferred with evident benefits in terms of submission performances.

Asynchronous Job Start

This is a configurable operating-mode of the WMProxy service. It has the goal to return control to the client as soon as the *jobStart* request has been accepted by WMProxy and the corresponding event has been logged to LB. All time-consuming actions needed to complete the processing of the request are performed "behind the scene" by the service. These actions encompass, e.g., the sub-jobs registration, the manipulation of their JDL descriptions and the creation of sub-job's reserved areas on the file system. If some of the processing fails, WMProxy logs the corresponding relevant event to LB so that user can check at any time the operation result by querying LB and eventually try to start again the job without loosing previous successful processing.

The asynchronous job start besides giving to the user a perception that service is much faster, has brought to a decoupling of the service modules performing actual

processing from the ones packaging messages for the interaction with clients that provides a significant level of flexibility for future improvements.

CONCLUSIONS AND FUTURE PLANS

In this paper we presented the details of the Workload Manager Proxy, the web services based interface to the gLite WMS, designed and implemented in the context of the EGEE project. The integration of new developments with existing Grid-components and Web service technologies allowed the transition of the WMS to the SOA paradigm as well as the addition of missing functionality.

This transition has appeared as an important design and implementation choice towards a future view of user needs, where functionalities are often seen as services. Performance improvements and new features availability have been reflected in positive feedback from "early adopters" together with a good perception of usability. First measures of bulk submission performances are very promising, in a context where further improvements seem to be reachable.

Deployment of the service in a production Grid infrastructure, such as the EGEE one, in the following months will provide further feedbacks of paramount importance for the evolution of the WMProxy service.

Future activities will focus on high-availability and scalability, where a clustered architecture could prove its effectiveness in such a direction. User authorization at the level of the single operations is already under implementation; the development of delivery plug-ins is also an interesting direction to investigate: WMProxy could then be seen as a proxy service for accessing different implementation of components providing functionality similar to those made available by the gLite Workload Management System.

The support for the Job Submission Description Language (JSDL), the new XML based language for job description that is currently a GGF recommendation, is for sure another priority together with the extension of the workflows management features.

Emerging, finalized WS* specifications will be also continuously monitored and adopted where deemed suitable.

ACKNOWLEDGEMENTS

This work has been supported by the EGEE project, which is funded by the European Commission's IST programme of the 6th Framework Programme.

REFERENCES

[1] G. Avellino et al., *The first deployment of workload management services on the EU DataGrid Testbed: feedback on design and implementation*.

[2] *Basic Profile Version 1.0*, The Web Services-Iteroperability Organization (WS-I) Web page, http://www.ws-i.org/Profiles/BasicProfile-1.0.html.

[3] *FastCGI: A High-Performance Gateway Interface*, http://www.fastcgi.com/devkit/doc/fastcgi-whitepaper/fastcgi.htm.

[4] GridSite Web page, http://www.gridsite.org.

[5] gSOAP: C/C++ Web Services and Clients Web page, http://www.cs.fsu.edu/~engelen/soap.html.

[6] WMProxy service WSDL file documentation, http://egee-jra1-wm.mi.infn.it/egee-jra1-wm/wmproxy.

[7] F. Pacini, *Job Description Language Attributes Specification*, EGEE-JRA1-TEC-590869-JDL-Attributes-v0-4, https://edms.cern.ch/document/590869/1.

[8] Condor Classified Advertisements (Classads) Web page, http://www.cs.wisc.edu/condor/classad.