Using CREAM and CEMonitor for job submission and management in the gLite middleware

C Aiftimiei[1,3], **P Andreetto**[1], **S Bertocco**[1], **S Dalla Fina**[1], **A Dorigo**[1], **E Frizziero**[1], **A Gianelle**[1], **M Marzolla**[1], **M Mazzucato**[1], **P Mendez Lorenzo**[2], **V Miccio**[2], **M Sgaravatto**[1], **S Traldi**[1], **L Zangrando**[1]

[1] INFN Padova, Via Marzolo 8, I-35131 Padova, Italy

[2] CERN, BAT. 28-1-019, 1211 Geneve, Switzerland

Abstract. In this paper we describe the use of CREAM and CEMonitor services for job submission and management within the gLite Grid middleware. Both CREAM and CEMonitor address one of the most fundamental operations of a Grid middleware, that is job submission and management. Specifically, CREAM is a job management service used for submitting, managing and monitoring computational jobs. CEMonitor is an event notification framework, which can be coupled with CREAM to provide the users with asynchronous job status change notifications. Both components have been integrated in the gLite Workload Management System by means of ICE (Interface to CREAM Environment). These software components have been released for production in the EGEE Grid infrastructure and, for what concerns the CEMonitor service, also in the OSG Grid. In this paper we report the current status of these services, the achieved results, and the issues that still have to be addressed.

1. Introduction

The job management component is a Grid component which is used to submit, cancel, and monitor jobs for execution on a suitable computational resource, called a Computing Element (CE). A CE is the interface to a large farm of computing hosts managed by a Local Resource Management System (LRMS), such as LSF or PBS. Moreover, a CE provides additional features to those of the underlying batch system, such as Grid-enabled user authentication and authorization, accounting, fault tolerance and improved performance and reliability.

In this paper we describe Computing Resource Execution and Management (CREAM) and CEMonitor, which are the components designed to manage a CE in the gLite Grid middleware. CREAM provides a simple, robust and lightweight service for job operations. It exposes an interface based on Web Services, which enables a high degree of interoperability with clients written in different programming languages. CEMonitor is a general-purpose asynchronous notification engine, which can be coupled with CREAM in order to notify clients when job status changes occur. Such a notification mechanism is particularly important for those clients which need to handle a large amount of jobs.

CREAM and CEMonitor are part of the gLite [1] middleware distribution and currently in production use within the EGEE Grid infrastructure [2]. CREAM can be used in stand-alone

[3] On leave from NIPNE-HH, Romania

mode, and users can interact directly with it through custom clients or using the provided C++-based command line tools. Furthermore, gLite users can submit jobs to CREAM through the gLite Workload Management System (WMS). For the latter case, a special component called Interface to Cream Environment (ICE) has been developed. ICE receives job submission and cancellation requests coming from a gLite WMS, and forwards these requests to CREAM. ICE then handles the entire lifetime of a job, including registering each status change to the gLite Logging and Bookkeeping (LB) service [3]. Note, however, that CREAM is mostly self-contained, with few dependencies on the gLite software components.

In this paper we describe the architecture of CREAM and CEMonitor. We describe their interfaces, and show how they have been integrated with the gLite infrastructure. Finally, we discuss the results of performance tests which have been recently performed in order to assess CREAM throughput.

2. The CREAM Service

CREAM is a job submission service: users can submit jobs, described via a classad-based Job Description Language (JDL) expression [4], to CREAM CEs. The JDL is a high-level notation based on Condor classified advertisements (classads) [5] for describing jobs and their requirements. CREAM supports the execution of batch (normal) and parallel (MPI) jobs. Normal jobs are single or multithreaded applications requiring one CPU to be executed; MPI jobs are parallel applications which usually require a larger number of CPUs to be executed, and which make use of the MPI library for interprocess communication. CREAM is a Java application which runs as an Axis servlet inside the Tomcat application server.

The CREAM interface exposes a set of operations which can be classified in three groups (see [6] for details).

The first group of operations (*Lease Management*) allows the user to define and manage leases associated with jobs. The lease mechanism has been implemented to ensure that all jobs get eventually managed, even if the CREAM service loses connection with the client application due to network partitioning. Each lease defines a time interval, and can be associated with a set of jobs. A lease can be renewed before its expiration; if a lease expires, all jobs associated with it are terminated and purged by CREAM.

The second group of operations (*Job Management*) is related with the core functionality of CREAM as a job management service. Operations are provided to create a new job, start execution of a job, suspend/resume or terminate a job. Moreover, the user can get the list of all owned jobs, and it is also possible to get the status of a set of jobs.

Finally, the third group of operations (*Service Management*) deals with the whole CREAM service. It consists of two operations, one for enabling/disabling new job submissions, and one for accessing general information about the service itself. Note that only users with administration privileges are allowed to enable/disable job submissions.

Authentication (implemented using a GSI based framework) is properly supported in all operations. Authorization on the CREAM service is also implemented, supporting both Virtual Organization (VO) based policies and policies specified in terms of individual Grid users. A Virtual Organization is a concept that supplies a context for operation of the Grid that can be used to associate users, their requests, and a set of resources. CREAM interacts with a VO Membership Service (VOMS) [7] service to manage VOs; VOMS is an attribute issuing service which allows high-level group and capability management and extraction of attributes based on the user's identity. VOMS attributes are typically embedded in the user's proxy certificate, enabling the client to authenticate as well as to provide VO membership and other evidence in a single operation.

3. The CEMonitor Service

CEMonitor provides an asynchronous event notification framework, which can be coupled with CREAM to notify the users when job status changes occur. Similarly to CREAM, CEMonitor is a Java application which runs in an Axis container within the Tomcat application server.

CEMonitor publishes information as *Topics*. For each Topic, CEMonitor maintains the list of *Events* to be notified to users. Topics can have three different levels of Visibility: *public*, meaning that everybody can receive events associated with the topic; *group*, meaning that only member of a specific VO can receive notifications; and *user*, meaning that only the user which created the Topic can receive notifications. Users can create *Subscriptions* for topics of interest. Each subscription has a unique ID, an expiration time and an update frequency f. CEMonitor checks every $1/f$ seconds whether there are new events for the topic associated to the subscription; if so, the events are sent to the subscribed users. Unless a subscription is explicitly renewed by its creator, it is removed after the expiration time and no more events will be notified.

Each Topic is produced by a corresponding *Sensor*. A Sensor is a component which is responsible for actually generating Events to be notified for a specific Topic. Sensors can be plugged at runtime: when a new Sensor is added, CEMonitor automatically generates the corresponding Topic so that users can subscribe. The most important Sensor we currently use is called *JobSensor*, which produces the events corresponding to CREAM job status changes. When CREAM detects that a job changes its status (for example, an IDLE job starts execution, thus becoming RUNNING, it notifies the JobSensor by sending a message on the network socket where the sensor is listening. Then, the JobSensor triggers a new notification which is eventually sent to all subscribed users.

Each Sensor can provide either asynchronous notifications to registered listeners, or can be queried synchronously. In both cases, Sensors support a list of so-called *Query Languages*. A Query Language is a notation (e.g., XPath, classad expressions and so on) which can be used to ask a Sensor to provide only Events satisfying a user-provided condition. When an Event satisfies a condition, CEMonitor triggers an *Action* on that event. In most cases, the Action simply instructs CEMonitor to send a notification to the user for that event. Of course, it is possible to extend CEMonitor with additional types of user-defined Actions. When registering for asynchronous notifications, the user passes a query expressed in one of the supported Query Languages as parameter. So, for that subscription, only events matching the query are notified.

Sensors support different *Dialects*. A Dialect is a specific output format which can be used to render Events. This means that a Sensor can publish information in different formats: for example, job status change information could be made available in Condor classad format [5], or in XML format. When a user subscribes to a Topic, she can also specify an appropriate Dialect for rendering the notifications. CEMonitor will then apply the correct rendering before sending the notifications.

Note that CEMonitor is a generic framework for information gathering and provisioning, and thus can also be used without being coupled with CREAM. For example in the context of the Open Science Grid (OSG) ReSS project is used to manage Grid resource information [8].

4. Integrating CREAM and CEMonitor in gLite

Users can interact directly with the CREAM services by means of a set of command line utilities which can be used to manage jobs by directly invoking CREAM operations. These command line tools are written in C++ using the gSOAP library [9].

CREAM functionality can also be used by the gLite WMS component [10]. This means that jobs submitted to the gLite WMS can be forwarded for execution on CREAM-based CEs.

The WMS comprises a set of Grid middleware components responsible for the distribution and management of tasks across Grid resources, in such a way that applications are conveniently, efficiently and effectively executed.

The CREAM–WMS integration is realized with a separate module, called ICE. ICE receives job submissions and other job management requests from the WMS component; it then uses the appropriate CREAM methods to perform the requested operation. Moreover, ICE is responsible for monitoring the state of submitted jobs and for taking the appropriate actions when the relevant job status changes are detected (e.g. the trigger of a possible resubmission if a Grid failure is detected).

Fig. 1 shows a schematic view of CREAM integration with the gLite WMS. gLite users submit jobs through the gLite User Interface (UI), which transfers jobs to a WMS, which then dispatches the request to the local ICE component. ICE interacts with CREAM via the legacy Web Service interface. Of course, users may also interact with CREAM directly (i.e., bypassing the gLite layer), by using the CREAM command line tools. In both cases the users must have a valid VOMS proxy credential. Other clients can concurrently access the Basic Execution Service (BES) interface to CREAM.

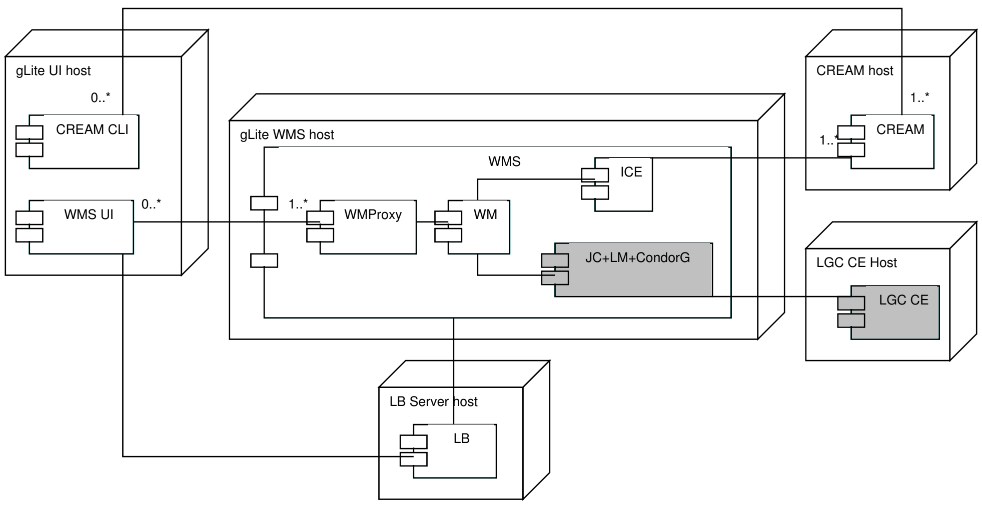

Figure 1. Job submission chain (simplified) in the gLite middleware

ICE obtains the state of a job in two different ways. The first one is by subscribing to a job status change notification service implemented by CEMonitor. CREAM notifies CEMonitor component about job state changes by using the shared, persistent CREAM backend. ICE subscribes to CEMonitor notifications, so it receives all status changes whenever they occur. As a fallback mechanism, ICE can explicitly query the CREAM service to check the status of "active" jobs for which it did not receive any notification for a configurable period of time. This mechanism guarantees that ICE knows the state of jobs (possibly with a coarse granularity) even if the CEMON service becomes unavailable. Job status change information is also sent to the LB [3] service, a distributed job tracking service.

All the components of the gLite middleware distribution (including CREAM and CEMonitor) are built using the ETICS Build and Test facility [11]. ETICS is an integrated system for the automated build, configuration, integration and testing of software. Using ETICS it is possible to integrate existing procedures, tools and resources in a coherent infrastructure, additionally providing an intuitive access point through a Web portal. The ETICS system allows developers to assemble multiple components, each one being developed independently, into a coherent software release. Each software component can use its own build method (e.g., Make for C/C++ code, Ant for Java code and so on), and ETICS provides a wrapper around that so that components or subsystems can be checked out and built using a common set of commands. The ETICS system can automatically produce and publish installation packages for the components it builds; multiple target platforms can also be handled.

CREAM and CEMonitor are included in the gLite 3.1 software distribution, which is provided as a set of different deployment modules (also called *node types*) that can be installed separately. CREAM and CEMonitor are installed and configured together as one of these modules, called `creamCE`. For what concerns the installation, the main supported platform, at present, is CERN Scientific Linux 4 (SLC4), 32-bit flavor; porting of gLite to CERN Scientific Linux 5 (64 bit) is underway. For the SLC4 platform, the gLite `creamCE` is available in RPM [12] format and the recommended installation method is via the gLite *yum* repository. For what concerns the configuration, there exists a manual configuration procedure but a gLite compliant configuration tool also exists. The tool adopted to configure gLite Grid Services is YAIM (YAIM Ain't an Installation Manager) [13]. YAIM provides simple configuration methods that can be used to set up uniform Grid sites. YAIM has been implemented as a set of bash scripts: it supports a component based model with a modularized structure including a YAIM core component, common to all the gLite middleware software, supplemented by component specific modules, all distributed as RPMs. For CREAM and CEMonitor appropriate plugins for YAIM were implemented in order to get a fully automated configuration procedure.

5. Performance tests

We evaluate the performance of the CREAM service in term of throughput (number of submitted jobs/s), comparing CREAM with the LCG-CE currently used in the gLite middleware. To do so, we submit 1000 identical jobs to an idle CE, using an otherwise identical infrastructure. The jobs are submitted using the credentials of four different users (each user submits 250 jobs).

All jobs are submitted using a WMS UI installed on the host `cream-15.pd.infn.it` located at INFN Padova. We always use the gLite WMS UI (see Fig. 1) for submissions to both CREAM and the LCG-CE (that is, we do not use direct CREAM submission). The UI transfers the jobs to the WMS host `devel19.cnaf.infn.it` located at INFN CNAF in Bologna. The WMS submits jobs through ICE to the CREAM service running on `cream-21.pd.infn.it` located at INFN Padova. The JobController+CondorG+LogMonitor components of the WMS submits jobs to a LCG-CE running on `cert-12.pd.infn.it`, also located at INFN Padova. Both CREAM and the LCG-CE are connected to the same (local) batch system running the LSF batch scheduler.

We are interested in examining the submission rate from ICE and JC/CondorG/LM to CREAM and LCG-CE respectively; this is an HB (Higher is Better) metric, as higher submission rate denotes better performance. To compute the submission rate we consider the time elapsed since the first job is dequeued by ICE or JC from their respective input queues, to the time the last job has been successfully transferred to the batch system . Note that we do not take into consideration the time needed to complete execution of the jobs.

In order to ensure that the transfer from WMS UI to the WMS is not the bottleneck in our tests, we execute the following steps:

(i) We switch off the ICE or JC component of the WMS;
(ii) We submit 1000 jobs from the WMS UI;
(iii) When all the jobs have been successfully transferred to the WMS node, we switch on ICE (or JC, depending on the kind of test we were performing). At this point ICE (or JC) finds all the jobs in its input queue, so what we measure here is the actual transfer rate from the WMS to the CE.

We analyze the impact of two factors on the submission throughput. The factors we consider are the following:

	Proxy Renewal	**Delegation**	**Submission Rate (jobs/sec)**	
			CREAM/ICE	LCG-CE/JC+LM
Test A	Disabled	Explicit	0.9624	0.3952
Test B	Disabled	Automatic	0.1660	0.3633
Test C	Enabled	Explicit	0.8976	0.3728
Test D	Enabled	Automatic	0.9191	0.3863

Table 1. Test results; better submission rates are shown underlined

- Use of an *automatic proxy renewal* mechanism vs *no proxy renewal.* The automatic proxy renewal mechanism is normally used for long-running jobs, to ensure that the credentials delegated to the CE are automatically refreshed before expiration. Automatic proxy renewal works by first having the user register her credentials to a so-called *MyProxy Server.* The gLite WMS receives a "fresh" proxy from the MyProxy server, and ICE or JC+CondorG are responsible for delegating the new credentials to the CE. We remark that no proxy is actually refreshed in our tests, since transfer of all jobs to the CE completes long before the user credentials expire. Nevertheless, the proxy renewal mechanism has an impact on the submission rate to CREAM via ICE, as will be explained later.
- Use of *automatic* vs *explicit delegation.* When *automatic* delegation is active, the WMS UI delegates a new proxy certificate to the WMS, which in turn delegates the proxy again to the CE, *for each job submitted to the CE.* Thus, a new delegation operation on the CE is executed before each submitted job. If *explicit* delegation is used, the user explicitly delegates a proxy before the first job is submitted, and uses the same delegation ID for all subsequent submissions. Thus, in this case only a single delegation operation is performed on the CE node.

We analyze four different scenarios with a total of 8 independent runs, corresponding to a 2^2 factorial design with two replications; each test has been repeated two times, and the average of the measured submission rates is considered.

Table 1 shows the submission rates for all the experiments. We observe that the submission rates from JC+CondorG+LM to the LCG-CE remain more or less the same across the different experiments. On the other hand, submission rates from ICE to the CREAM CE are higher in three of our experiments, but incur a significant penalty in Test B.

The reason for this is in the different way in which CREAM/ICE and LCG-CE/JC+CondorG+LM implement the transfer of user credentials from the WMS to the CE node. CREAM exposes a delegation port-type to allow clients to securely delegate their credentials to the CE. The delegation operation involves the creation on the server side of a public/private key pair, which takes a considerable amount of time. Explicit delegation (Test A and C) allows ICE to delegate only once for each user: in our tests, as we are submitting 250 jobs for each of 4 different users, only four delegation operations are performed, and this causes a significant improvement of the submission rate.

The JC+CondorG+LM does not implement a proper delegation operation, but *for each job* transfers the user credentials to the LCG-CE using a more lightweight mechanism. This explains why the submission rate achieved by LCG-CE/JC+CondorG+LM is more or less independent from the delegation mechanism used (automatic or explicit). The lack of delegation on the LCG-CE was one of the reasons why CREAM was developed, as credential transfer without proper delegation is no longer considered acceptable.

In Test D we have automatic delegation together with proxy renewal. This implies that *all* delegated user proxies are automatically renewed. Note that if the same user performs two delegations, the delegated credentials will expire on different times, and thus in general should be treated separately. However, if the proxy renewal mechanism is active, all delegations will be renewed before expiration, so from the user point of view all her credentials have duration equal to the duration of the proxy renewal mechanism. For this reason, in situations like Test D, ICE considers all proxies “equivalent” by performing a single delegation operation to CREAM for each user which requested automatic credentials renewal.

We also performed a separate test to assess the reliability of the CREAM service. More specifically, we tested whether CREAM was able to ensure that all submitted jobs were eventually executed. To so do, we submitted a continuous flow of jobs to 21 CREAM CEs through the gLite WMS. 14 of the CEs were at INFN Padova, Italy, and the remaining 7 CEs were at INFN CNAF in Bologna, Italy. We submitted a collection of 40 jobs every minute to a gLite WMS for all the 5 days of duration of the test. The duration of each job was 5 minutes, and the submission rate and job duration were calibrated to ensure that a more or less constant number of jobs were active (i.e., not in a terminal state) in the CEs at any given time. We enabled the proxy renewal mechanism (the user proxy used for submitting each job had an initial duration of about 5 hours). Automatic resubmission of failed jobs back to the WMS was enabled, so that CREAM was able to resubmit jobs failed for external reasons (e.g., a worker node crashing) back to the WMS. The WMS then rescheduled resubmitted jobs to a different CREAM CE up to a maximum number of times, after which the job is set to the Aborted state and is no longer resubmitted. The test results are as follows: 99.2% of jobs terminated successfully (that is, they reached the Done-OK state). No jobs reached the Aborted state, and the remaining 0.8% of jobs did not finish execution due to a problem in the Torque batch system which has been fixed since then. 1.60% of the jobs were resubmitted to the WMS.

Finally, the CREAM based CE was also tested and used for real production activities. To assess the performance and the reliability of CREAM, and in particular to verify its usability in production environments, the Alice LHC experiment [14] performed some tests which took place during the summer of 2008. About 55000 standard production Alice jobs, each one lasting about 12 hours, were submitted on a CREAM based CE at the FZK[4] Tier-1 center. The CREAM service showed a remarkable stability: no failures were seen and no manual interventions were needed during the whole test period.

After this first successful assessment, the submission to CREAM based CREAM CEs has been fully integrated in the Alien Alice software environment. Alice jobs are currently being submitted in about a dozen of CREAM CEs deployed in several sites of the EGEE Grid.

6. Conclusions

In this paper we described the general architecture of CREAM, a Java-based Grid CE service. CREAM uses a Web Service interface to provide its functions via an interface based on Web Services, a lightweight implementation and a rich set of features. It is being integrated with the Grid infrastructure, in particular with the WMS subsystem, by means of a glue component called ICE. Moreover, CREAM is being extended with a BES/Job Submission Description Language (JSDL) compliant interface.

More detailed information, including installation instructions, interface specification and usage manual for CREAM can be found on the Web page [15].

7. Acknowledgments

EGEE-3 is a project funded by the European Union under contract INFSO-RI-222667.

[4] Forschungszentrum Karlsruhe, `http://www.fzk.de/`

References

[1] Laure E, Fisher S M, Frohner Á, Grandi C, Kunszt P, Krenek A, Mulmo O, Pacini F, Prelz F, White J, Barroso M, Buncic P, Hemmer F, Di Meglio A and Edlund A 2006 *Computational Methods in Science and Technology* **12** 33–45

[2] Enabling Grid for E-sciencE (EGEE) project web site `http://www.eu-egee.org/`

[3] Kouřil D *et al.* 2004 *Proceedings of CHEP'04* (Interlaken, Switzerland)

[4] Sgaravatto M 2005 *CREAM Job Description Language Attributes Specification for the EGEE Middleware* document Identifier EGEE-JRA1-TEC-592336, Available online at `https://edms.cern.ch/document/592336`

[5] Raman R 2001 *Matchmaking Frameworks for Distributed Resource Management* Ph.D. thesis

[6] Aiftimiei C, Andreetto P, Bertocco S, Fina S D, Dorigo A, Frizziero E, Gianelle A, Marzolla M, Mazzucato M, Sgaravatto M, Traldi S and Zangrando L 2009 Design and implementation of the glite cream job management service Technical Note INFN/TC_09/3 Istituto Nazionale di Fisica Nucleare

[7] Alfieri R, Cecchini R, Ciaschini V, dell'Agnello L, Frohner Á, Lőentey K and Spataro F 2005 *Future Generation Computer Systems* **21** 549–558 ISSN 0167-739X

[8] Garzoglio G, Levshina T, Mhashilkar P and Timm S 2009 *Grid Computing, International Symposium on Grid Computing (ISGC 2007)* ed Lin S C and Yen E (Springer) pp 89–98

[9] van Engelen R gSOAP 2.7.6 user guide 29 Dec. 2005

[10] Andreetto P *et al.* 2004 *Proceedings of CHEP'04* (Interlaken, Switzerland)

[11] Bégin M E, Sancho G D A, Meglio A D, Ferro E, Ronchieri E, Selmi M and Zurek M 2006 *RISE* (*Lecture Notes in Computer Science* vol 4401) ed Guelfi N and Buchs D (Springer) pp 81–97 ISBN 978-3-540-71875-8

[12] Foster-Johnson E 2003 *Red Hat RPM Guide* 1st ed (Red Hat) ISBN 978-0764549656

[13] YAIM Home Page `http://yaim.info/`

[14] ALICE–A Large Ion Collider Experiment at CERN LHC `http://aliceinfo.cern.ch/`

[15] CREAM home page `http://grid.pd.infn.it/cream`