ISTITUTO NAZIONALE DI FISICA NUCLEARE
Sezione di Padova

INFN/TC-09/3
May 5, 2009

Design and Implementation of the gLite CREAM Job Management Service

Cristina Aiftimiei, Paolo Andreetto, Sara Bertocco, Simone Dalla Fina,
Alvise Dorigo, Eric Frizziero, Alessio Gianelle, Moreno Marzolla, Mirco Mazzucato,
Massimo Sgaravatto, Sergio Traldi, Luigi Zangrando
INFN, Sezione di Padova, via Marzolo 8, I-35131 Padova, Italy

Abstract

Job execution and management is one of the most important functionality provided by every modern Grid middleware. In this paper we describe how the problem of job management has been addressed in the gLite middleware by means of the CREAM and CEMonitor services. CREAM (Computing Resource Execution and Management) provides a job execution and management capability for Grid systems, while CEMonitor is a general-purpose asynchronous event notification framework. Both services expose a Web Service interface allowing conforming clients to submit, manage and monitor computational jobs to a Local Resource Management System.

PACS: 89.20.Ff Computer Science and Technology

Published by **SIS-Pubblicazioni**
Laboratori Nazionali di Frascati

1 Introduction

Grid middleware distributions are often large software artifacts, which include a set of components each providing a basic functionality. Such capabilities include (but are not limited to) data storage, authentication and authorization, resource monitoring, and job management. The job management component is used to submit, cancel, and monitor jobs which are executed on a suitable computational resource, usually referred as a Computing Element (CE). A CE is the interface to a usually large farm of computing hosts managed by a Local Resource Management System (LRMS), such as LSF or PBS. Moreover, a CE implements additional features with respect to the ones provided by 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 the architecture of Computing Resource Execution and Management (CREAM), a system designed to efficiently manage a CE in a Grid environment. 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: currently Java and C++ clients are provided, but it is possible to use any language with a Web Service framework. CREAM itself is written in Java, and runs as an extension of a Java-Axis servlet inside the Apache Tomcat application server [30].

As stated before, it is important for users to be able to monitor the status of their jobs. This means checking whether the job is queued, running, or finished; moreover, extended status information (such as exit code, failure reason and so on) must be obtained from the job management service. While CREAM provides an explicit operation for querying the status of a set of jobs, it is possible to use a separate notification service in order to be notified when a job changes its status. This service is provided by CEMonitor, which is a general-purpose asynchronous notification engine. CEMonitor can be used by CREAM to notify the user about job status changes. This feature is particularly important for specialized CREAM clients which need to handle a large amount of jobs. In these cases, CEMonitor makes the expensive polling operations unnecessary, thus reducing the load on CREAM and increasing the overall responsiveness.

CREAM and CEMonitor are part of the gLite [23] middleware distribution and currently in production use within the EGEE Grid infrastructure [2]. Users can install CREAM in stand-alone mode, and interact directly with it through custom clients or using the provided C++-based command line tools. Moreover, gLite users can transparently 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 devel-

oped. 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 [22]. Note, however, that CREAM is mostly self-contained, with few dependencies on the gLite software components.

1.1 Related Works

The problem of job management is addressed by any Grid system. Different job management services have been developed starting from different requirements; furthermore, they must take into account the specific features of the middleware they belong to.

The UNICORE (Uniform Interface to Computing Resources) [13] system was initially developed to allow German supercomputer centers to provide seamless and secure access to their computational resources. Architecturally, UNICORE is a three-tier system. The first tier is made of clients, which submit requests to the second tier (server level). The server level of UNICORE consists of a Gateway which authenticates requests from UNICORE clients and forwards them to a Network Job Supervisor (NJS) for further processing. The NJS maps the abstract requests into concrete jobs or actions which are performed by the target system. Sub-jobs that have to be run at a different site are transferred to this site's gateway for subsequent processing by the peer NJS. The third tier of the architecture is the target host which executes the incarnated user jobs or system functions.

The Advanced Resource Connector (ARC) [12] is a Grid middleware developed by the NorduGrid collaboration. ARC is based on the Globus Toolkit[1], and basically consists of three fundamental components: the *Computing Service* which represents the interface to a computing resource (generally a cluster of computers); the *Information System* which is a distributed database maintaining a list of know resources; and a *Brokering Client* which allows resource discovery and is able to distribute the workload across the Grid.

The Globus Toolkit provides both a suite of services to submit, monitor, and cancel jobs on Grid computing resources. GRAM4 refers to the Web Service implementation of such services [15]. GRAM4 includes a set of WSRF-compliant Web Services [16] to locate, submit, monitor, and cancel jobs on Grid computing resources. GRAM4 is not a job scheduler, but a set of services and clients for communicating with different batch/cluster job schedulers using a common protocol. GRAM4 combines job-management services and local system adapters with other service components of the Globus Toolkit in order to support job execution with coordinated file staging.

[1] Globus and Globus Toolkit are trademarks of the University of Chicago

Initially, the job management service of the gLite middleware was implemented by the legacy LGC-CE [9], which is based on the pre-Web Service version of GRAM. The development of CREAM was motivated by some shortcomings of the existing solutions. It was necessary to address scalability and performance problems with the existing solutions. Furthermore, with the consolidation of open Web standards it was necessary to develop a new, cross-platform Web Service-based CE.

1.2 Organization of this paper

This paper is organized as follows. In Section 2 we give a general overview on how job management is implemented in the gLite middleware. Then, in Section 3 we restrict our attention on the CREAM and CEMonitor services, which are the final part of the job management chain in gLite. Sections 4 and 5 describe the architecture of CREAM and CEMonitor respectively. In Section 6 we describe the interactions with CREAM and CEMonitor which are necessary to handle the typical job submission sequence. Section 7 describes how the components are built and deployed in the production infrastructure. Section 8 describes some performance results. Finally, conclusions and future works are discussed in Section 9.

2 Job Management in the gLite Middleware

In this section we give a brief introduction to the job management architecture of the gLite middleware. The interested reader is referred to [23,9] for a more complete description.

Fig. 1 shows the main components involved in the gLite job submission chain. We will consider job submission to the CREAM CE only. The components shown in gray in the figure–namely JobController+LogMonitor+CondorG and LCG-CE–are those responsible for job management through the legacy LCG-CE, and will not be described in this paper.

There are two entry points for job management requests: the gLite WMS User Interface (UI) and the CREAM UI. Both include a set of command line tools which can be used to submit, cancel and query the status of jobs. In gLite, jobs are described using the Job Description Language (JDL) notation, which is a textual notation based on Condor classads [26].

The CREAM UI is used to interact directly with a specific CREAM CE. It is a set of command line tools, written in C++ using the gSoap engine [32]. The CREAM CLI provides a set of commands to invoke the Web Services operations exposed by CREAM (see Table 1 on Section 4 for the list of available operations). The user can submit, cancel, and query the status of a job on a CREAM server.

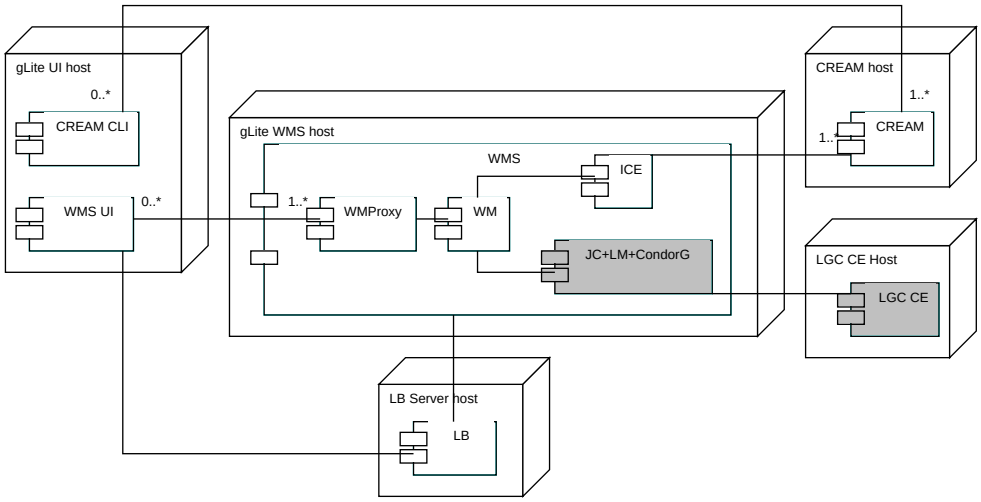

Figure 1: Job submission chain (simplified) in the gLite middleware

On the other hand, the gLite WMS UI allows the user to submit and monitor jobs through the gLite Workload Management System (WMS) [5]. The WMS is responsible for the distribution and management of tasks across Grid resources (in particular Computing Elements), in such a way that applications are efficiently executed. Job management through the WMS provides many benefits compared to direct job submission to the CE:

- The WMS can manage multiple CEs, and is able to forward jobs to the one which better satisfies a set of requirements which can be specified as part of the job description;
- The WMS can be instructed to handle job failures: if a job aborts due to problems related with the execution host (e.g. host misconfiguration) the WMS can automatically resubmit it to a different CE;
- The WMS provides a global job tracking facility using the LB service;
- The WMS supports complex job types (job collections, job with dependencies) which can not be handled directly by the CEs.

Note that there is a many to many relationship between the gLite WMS UI and the WMS, that is, multiple User Interfaces can submit to the same WMS, and multiple WMS can be associated to the same WMS UI.

The WMS exposes a Web Service interface which is implemented by the WMProxy component. The core of the WMS is the Workload Manager (WM), whose purpose is to

accept and satisfy requests for job management. For job submission requests, the WM tries to locate an appropriate resource (CE) where the job can be executed. The decision of which resources should be used is the outcome of the matchmaking process between the requests and the available resources. The user can specify a set of *requirements* in the job description. These requirements represent a set of constraints which the WM tries to satisfy when selecting the CE where the job will be executed.

Currently, the gLite WMS can submit jobs to CREAM as well as to the legacy LCG-CE. Each CE is uniquely identified by a URI called *ce-id*. Interaction with the LCG-CE is handled by the Job Controller/Log Monitor/CondorG (JC/LM/CondorG) modules within the WMS. In the case of submission to CREAM-based CEs, jobs are managed by a different module, called ICE. ICE receives job submissions and other job management requests from the WM component of the WMS through a simple messaging system based on local files. ICE then uses the operations of the CREAM interface to perform the requested operation. Moreover, it is responsible for monitoring the state of submitted jobs and for taking the appropriate actions when job status changes are detected (e.g. to trigger a possible resubmission if a Grid failure is detected).

ICE can obtain the state of a job in two different ways. The first one is by subscribing to a job status change notification service implemented by a separate component called CEMonitor (more details in Section 5). CEMonitor [10] is a general purpose event notification framework. CREAM notifies the 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 also poll 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 even if the CEMonitor service becomes unavailable or has not been installed.

The LB service [22] is used by the WMS to store various information on running jobs, and provide the user with an overall view on the job state. The service collects events in a non blocking asynchronous way, and this information can be used to compute the job state. LB is also used to store events such as the transfer of jobs from one component to another one (e.g., from the WMproxy to the WM): in this way, the user knows the location of each job. The job status information gathered by the LB is made available through the gLite UI commands. Note that in case of direct submissions through the CREAM UI, the LB service is not used; however, CREAM itself provides the *JobInfo* operation for reporting detailed job status information.

3 CREAM and CEMonitor

CREAM and CEMonitor are both available through Web Service interfaces. CREAM is intended to offer job management facilities to the widest possible range of consumers. This includes not only other components of the same middleware stack, but also single users and other heterogeneous services. Thus, we need a mechanism that lets potential users to be as free as possible in using their own tools and languages to interface to CREAM and CEMonitor. The Web Services technology offers all the interoperability characteristics that are needed to fulfill the above requirements.

3.1 Deployment

Fig. 2 shows the typical deployment of a Computing Element based on CREAM and CEMonitor. Both applications run as Java-Axis servlets [7] in the Tomcat application server [30]. Requests to CREAM and CEMonitor traverse a pipeline of additional components which take care of authorization issues; one of these component is the *Authorization Framework*, which is an Axis plugin for validating and authorizing the requests received by the services (more details on the security infrastructure will be given shortly).

CREAM uses an external relational database server to store its internal state. This improves fault-tolerance as it guarantees that this information is preserved across restarts of CREAM. Moreover, the use of a SQL database improves responsiveness of the service while performing complex queries which are needed by the normal CREAM operations, such as getting the list of jobs associated with a specific user. The database is accessed through the JDBC interface; in the gLite deployment we are using MySQL [11], but any database accessible through JDBC is supported. Note that the database server can be installed on a dedicated host, as shown in Fig. 2, or can share the same machine as CREAM and CEMonitor.

CREAM interacts with CEMonitor [10] to provide an asynchronous job status notification service. For each job status change, CREAM notifies CEMonitor, which in turn check whether there are subscriptions registered for that notification. If so, the notification is sent to the user which requested that (more details will be given in Section 5).

CREAM can be associated to multiple batch queues (note the one-to-many association shown in Fig. 2). CREAM submits requests to the LRMS through Batch-system Local ASCII Helper (BLAH) [25], an abstraction layer for the underlying LRMS. BLAH, in turn, interacts with the client-side LRMS environment, which might consist of a set of command line tools which interact with the server-side LRMS.

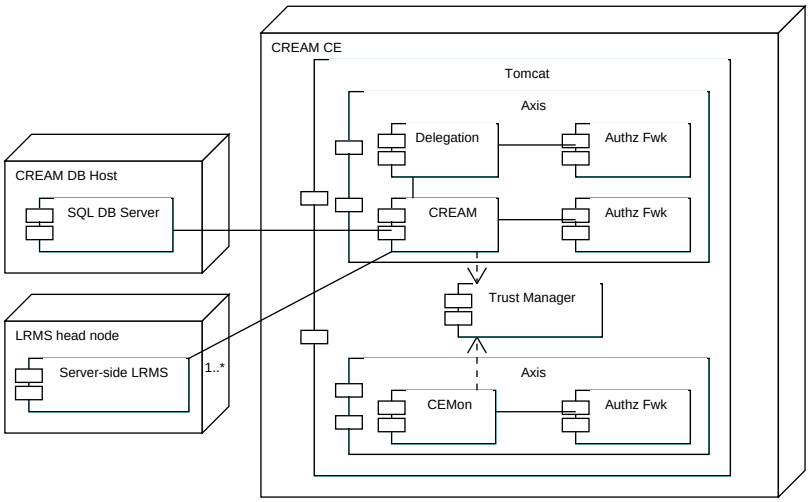

Figure 2: Typical deployment of a CREAM service

3.2 Security

The Grid is a large collaborative resource-sharing environment. Users and services cross the boundaries of their respective organizations and thus resources can be accessed by entities belonging to several different institutions. In such a scenario, security issues are of particular relevance. There exists a wide range of authentication and authorization mechanisms, but Grid security requires some extra features: access policies are defined both at the level of Virtual Organizations (VOs) and at the level of single resource owners. Both these aspects must be taken into account. Moreover, as we will see in the following, Grid services have to face the problem of dealing with the delegation of certificates and the mapping of Grid credentials into local batch system credentials.

Trust Manager The Trust Manager is the component responsible for carrying out authentication operations. It is external to CREAM and CEMonitor, and is an implementation of the J2EE security specifications [28]. Authentication is based on Public Key Infrastructure (PKI). Each user (and Grid service) wishing to access CREAM or CEMonitor is required to present an X.509 format certificate [20]. These certificates are issued by trusted entities, the Certificate Authorities (CA). The role of a CA is to guarantee the identity of a user. This is achieved by issuing an electronic document (the certificate) that contains the information about the user and is digitally signed by the CA with its private key. An authentication manager, such as the Trust Manager, can verify the user identity by decrypting the hash of the certificate with the CA public key. This ensures that the certificate was issued by that specific CA. The Trust Manager can then access the user

data contained in the certificate and verify the user identity. One interesting challenge in a Grid environment is the so-called *proxy delegation*. It may be necessary for a job running on a CE to perform some operations on behalf of the user owning the job. Those operations might require proper authentication and authorization support. For example, we may consider the case where a job running on a CE has to access a Storage Element (SE) to retrieve or upload some data. This aim is achieved in the Trust Manager using *proxy certificates*. RFC3820 proxy certificates are an extension of X.509 certificates [31]. The generation of a proxy certificate is as follows. If a user wants to delegate her credential to CREAM, she has to contact the *delegation Port-type* of the service. CREAM creates a public-private key pair and uses it to generate a Certificate Sign Request (CSR). This is a certificate that has to be signed by the user with her private key. The signed certificate is then sent back to CREAM. This procedure is similar to the generation of a valid certificate by a CA and, in fact, in this context the user acts like a CA. The certificate generated so far is then combined with the user certificate, thus forming a chain of certificates. The service that examines the proxy certificate can then verify the identity of the user that delegated its credentials by unfolding this chain of certificates. Every certificate in the chain is used to verify the authenticity of the certificate at the previous level in the chain. At the last step, a CA certificate states the identity of the user that first issues the delegated proxy.

Authorization Framework The aim of the authorization process is to check whether an authenticated user has the rights to access services and resources and to perform certain tasks. The decision is taken on the basis of policies that can be either local or decided at the VO level. Administrators need a tool that allows them to easily configure the authorization system in order to combine and integrate both these policies. For this reason, CREAM adopts a framework that provides a light-weight, configurable, and easily deployable policy-engine-chaining infrastructure for enforcing, retrieving, evaluating and combining policies locally at the individual resource sites. The framework provides a way to invoke a chain of policy engines and get a decision result about the authorization of a user. The policy engines are divided in two types, depending on their functionality. They can be plugged into the framework in order to form a chain of policy engines as selected by the administrator in order to let him set up a complete authorization system. A policy engine may be either a Policy Information Point (PIP) or a Policy Decision Point (PDP). PIPs collect and verify assertions and capabilities associated with the user, checking her role, group and VO attributes. PDPs may use the information retrieved by a PIP to decide whether the user is allowed to perform the requested action, whether further evaluation is needed, or whether the evaluation should be interrupted and the user access denied.

In CREAM both VO and "ban/allow" based authorizations are supported. In the former scenario, implemented via the VOMS PDP, the administrator can specify authorization policies based on the VOs the jobs' owners belong to (or on particular VO attributes). In the latter case the administrator of the CREAM-based CE can explicitly list all the Grid users (identified by their X.509 Distinguished Names) authorized to access CREAM services. For what concerns authorization on job operations, by default each user can manage (e.g. cancel, suspend, etc.) only her own jobs. However, the CREAM administrator can define specific "super-users" who are empowered to manage also jobs submitted by other users.

Credential Mapping The execution of user jobs in a Grid environment requires isolation mechanisms for both applications (to protect these applications from each other) and resource owners (to control the behavior of these arbitrary applications). In the absence of solutions based on the virtualization of resources (VM), CREAM implements isolation via local credential mapping, exploiting traditional Unix-level security mechanisms like a separate user account per Grid user or per job. This Unix domain isolation is implemented in the form of the `gLExec` system [19], a sudo-style program which allows the execution of the user's job with local credentials derived from the user's identity and any accompanying authorization assertions. This relation between the Grid credentials and the local Unix accounts and groups is determined by the Local Credential MAPping Service (LCMAPS) [24]. `gLExec` also uses the Local Centre Authorization Service (LCAS) to verify the user proxy, to check if the user has the proper authorization to use the `gLExec` service, and to check if the target executable has been properly "enabled" by the resource owner.

4 The CREAM service

The main functionality of CREAM is job management. Users submit jobs described as a JDL expression [27] representing a job, and CREAM executes it on an underlying LRMS (batch system). The JDL is a high-level, user-oriented notation based on Condor classified advertisements (classads) [26] for describing jobs and their requirements. CREAM uses a JDL dialect which is very similar to the one used to describe jobs in the gLite WMS. There are however some differences between the CREAM and WMS JDL, which are motivated by the different role of the job execution and workload management services. As described in Section 2, the gLite WMS receives job submission requests which possibly include a set of user-defined requirements, which are used by the WM to select the CE where the job is executed. Of course, once the selection is done, there

is no need for the CE to further process the job requirements as they are no longer relevant. Similarly, there are other kind of information which only make sense for the CREAM JDL, and not for the WMS JDL.

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.

Applications executed by CREAM might need a set of input data files to process, and might produce a set of output data files. The set of input files is called the InputSandBox (ISB), while the set of files produced by the application is called the OutputSandBox (OSB). CREAM transfers the ISB to the executing node from the client node and/or from Grid storage servers to the execution node. The ISB is staged in before the job is allowed to start. Similarly, files belonging to the OSB are automatically transferred out of the execution node when the job terminates.

As an example, consider the following JDL processed by CREAM:

```
[
  Type = "job";
  JobType = "normal";
  Executable = "/sw/command";
  Arguments = "60";
  StdOutput = "sim.out";
  StdError = "sim.err";
  OutputSandbox = {
    "sim.err",
    "sim.out"
  };
  OutputSandboxBaseDestURI = "gsiftp://se1.pd.infn.it:5432/tmp";
  InputSandbox = {
    "file:///home/user/file1",
    "gsiftp://se1.pd.infn.it:1234/data/file2",
    "/home/user/file3",
    "file4"
  };
  InputSandboxBaseURI = "gsiftp://se2.cern.ch:5678/tmp";
]
```

With this JDL a *normal* (batch) job will be submitted. Besides the specification of the executable `/sw/command` (which must already be available in the file system of the executing node, since it is not listed in the ISB), and of the standard output/error files, it is specified that the files `file1`, `file2`, `file3`, `file4` will have to be staged on the executing node as follows:

- `file1` and `file3` will be copied from the client UI file system
- `file2` will be copied from the specified GridFTP server (`gsiftp://se1.pd.infn.it:1234/data/file2`)

- `file4` will be copied from the GridFTP server specified as `InputSandboxBaseURI` (`gsiftp://se2.cern.ch:5678/tmp`)

It is also specified that the files `sim.err` and `sim.out` (specified as `OutputSandbox`) must be automatically uploaded into `gsiftp://se1.pd.infn.it:5432/tmp` when the job completes its execution.

The pre- and post-staging of data is handled by a shell script, called Job Wrapper (JW), which is what is actually sent for execution on the LRMS. As the name suggests, the script "wraps" the executable by taking care of fetching external data, then calling the executable and finally putting the output data to the correct remote locations. The JW is assembled by CREAM according to the JDL and sent to the LRMS.

Other typical job management operations (job cancellation, job status with different levels of verbosity and filtering, job listing, job purging) are supported. Moreover users are allowed to suspend and resume jobs submitted to CREAM-based CEs, provided that the underlying LRMS supports this feature.

For what concerns security, authentication (implemented using a GSI based framework [15]) is properly supported in all operations. Authorization on the CREAM service is also implemented, supporting both 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 the VO Membership Service (VOMS) [4] 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.

Fig. 3 shows the (simplified) internal structure of the CREAM service. CREAM exposes two different Web Service interfaces. The operations of the legacy CREAM interface are listed in Table 1.

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

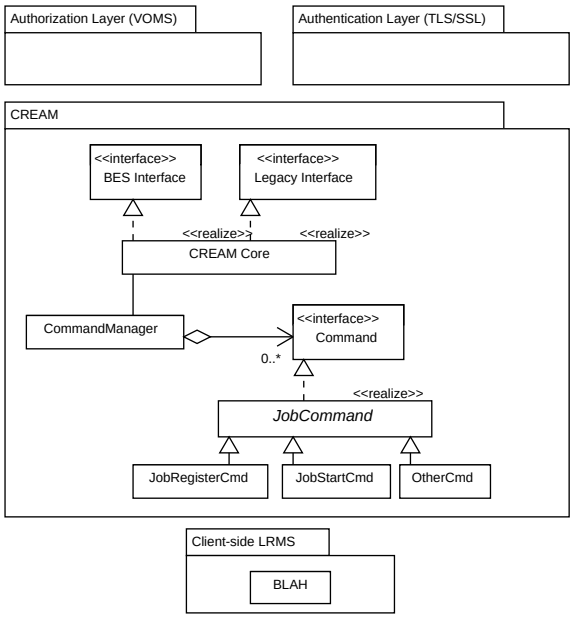

Figure 3: CREAM internal architecture

can get the list of all owned jobs, and it is also possible to get the status of a set of jobs. The CREAM job state model is shown in Fig. 4, and job states are described in Table 2.

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.

Recently we implemented an additional interface to the CREAM service, compliant with the Basic Execution Service (BES) specification. BES [14] defines a standard interface for execution services provided by different Grid systems. The aim of BES is to favor interoperability of computing elements between different Grids: the same BES-enabled CE can be "plugged" into any compliant infrastructure; moreover, sharing of resources between different Grids is possible. BES defines basic operations for job submission and management. More specifically, the BES specification defines two Web Services *port-types*: *BES-Factory*, containing operations for creating, monitoring and controlling sets of jobs, and *BES-Management*, which allows clients to monitor the details of and control the BES itself. The Port-types and associated operations are shown in Table 3.

BES uses the Job Submission Description Language (JSDL) [6] as the notation for describing computational jobs. The legacy CREAM interface was defined before BES was available, and also provides additional methods which are not provided by BES (notably, the possibility to renew a user proxy certificate, which is useful to avoid user proxy

Lease Management Operations	
SetLease	Creates a new lease, or renews an existing lease
GetLease	Gets information on a lease with given ID
JobSetLeaseId	Associates a lease with a job
GetLeaseList	Gets the list of all active leases
DeleteLease	Deletes a lease, and purge all associated jobs
Job Management Operations	
JobRegister	Registers a new job for future execution
JobStart	Starts execution of a registered job
JobCancel	Terminates a job
JobPurge	Purges all information of a job
JobSuspend	Suspends execution of a running job
JobResume	Resumes execution of a suspended job
JobStatus	Gets the status of a job
JobInfo	Gets detailed information about a job
JobList	Gets the list of all active jobs
Service Management Operations	
acceptNewJobSubmissions	Enables/disables new job submissions
getServiceInfo	Gets general information about the service

Table 1: CREAM interface operations

expiration while a job is running).

CREAM has been developed around an internal core, which is a generic command executor. The core accepts abstract commands which are enqueued and executed by a pool of threads. It is possible to customize the core by defining concrete implementations of the abstract command interface. Two kind of commands can be defined: *synchronous* and *asynchronous*. Synchronous commands must be executed immediately upon receipt, while asynchronous command execution can be deferred at a later time. Moreover, it is possible to define *sequential* or *parallel* commands. When a parallel command is being executed, other commands (parallel or sequential) can be concurrently executed by other threads in the pool. When a sequential command is being executed, no other commands operating on the same job are executed by any other thread, until the sequential command terminates execution. The job management interfaces (both the BES and the legacy one) instantiate the correct command type to execute the operations requested by the users.

When job submissions arrive through the gLite WMS, it is essential that all jobs submitted to CREAM eventually reach a terminal state (and thus eventually get purged from the CREAM server). The gLite WMS has been augmented with an additional com-

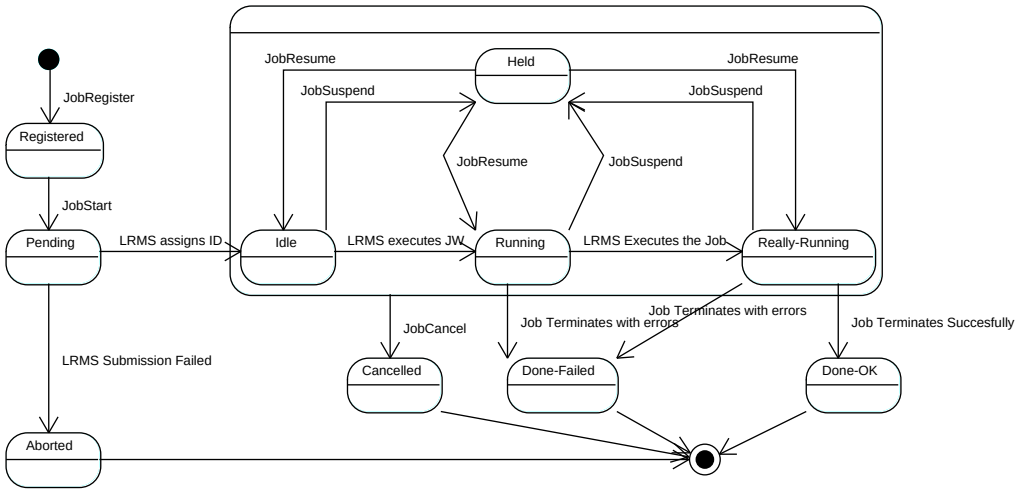

Figure 4: CREAM job states

ponent, ICE, which is responsible for interacting with CREAM. ICE and CREAM use a lease-based protocol to ensure that all jobs get eventually purged by CREAM even if it loses contact with the ICE client, e.g. due to network partitioning. Basically, each job submitted through ICE has an associated *lease time*, which must be periodically renewed using the *JobLease* CREAM operation. ICE is responsible for periodically renewing the leases associated to active jobs, i.e. jobs which are not terminated yet. Should a lease expire before the actual termination of a job, CREAM will purge all jobs associated with that lease and free all the CE resources used by them.

CREAM can be seen as an abstraction layer on top of an LRMS (batch system), which extends the LRMS capabilities with an additional level of security, reliability, and integration with a Grid infrastructure. CREAM supports different batch systems through the idea of *LRMS connectors*. An LRMS connector is an interface for a generic batch system. Currently, CREAM supports all the batch systems supported by BLAH [25] through a specific instance of LRMS connector called the *BLAH connector module*: at the time of writing BLAH supports LSF, PBS/Torque, and Condor [29]; Sun Grid Engine (SGE) support is currently being implemented as well. It is also possible to create other, ad-hoc connectors to interact with other types of batch systems. Note that a single instance of CREAM can provide access to multiple underlying LRMS.

5 The CEMonitor Service

The purpose of CEMonitor is to provide an asynchronous event notification framework, which can be coupled with CREAM to notify the users when job status changes occur.

Registered	The job has been submitted to CREAM with the *JobRegister* operation
Pending	The user invoked the *JobStart* operation to start the job execution
Idle	The LRMS (batch system) accepted the job for execution. The job is now in the LRMS queue
Running	The Job Wrapper is being executed
Really-Running	The actual user job is being executed
Held	The job has been suspended, e.g. because the user issued the *JobSuspend* operation. The job can be resumed in its previous state with the *JobResume* operation
Done-OK	The job terminated correctly
Done-Failed	The job terminated with errors
Cancelled	The job has been cancelled, e.g. because the user invoked the *JobCancel* operation to terminate it
Aborted	Submission to the LRMS failed

Table 2: Description of the CREAM job states

Figure 5 shows the internal structure of the CEMonitor service. Similarly to CREAM, CEMonitor is a Java application which runs in an Axis container within the Tomcat application server. CEMonitor uses the same authentication/authorization mechanisms as CREAM, which has been already discussed in Section 3.

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

BES-Management Port-type	
StartAcceptingNewActivities	Administrative operation: requests that the BES service start accepting new activities
StopAcceptingNewActivities	Administrative operation: requests that the BES service stop accepting new activities
BES-Factory Port-type	
CreateActivity	Requests the creation of a new activity; in general, this operation performs the submission of a new computational job, which is immediately started
GetActivityStatuses	Requests the status of a set of activities
TerminateActivities	Requests termination of a set of activities
GetActivityDocuments	Requests the JSDL document for a set of activities
GetFactoryAttributeDocument	Requests the XML document containing the properties of this BES service

Table 3: BES Port-Types and Operations

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 with the *Subscribe* CEMonitor operation (see Table 4), 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 [26], 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.

We show in Fig. 6 an example of job status change notification. The notification is in Condor classad format, and contains several variables with their associated values. `CREAM_JOB_ID` is the ID of the job which changed status; `CREAM_URL` is the endpoint of

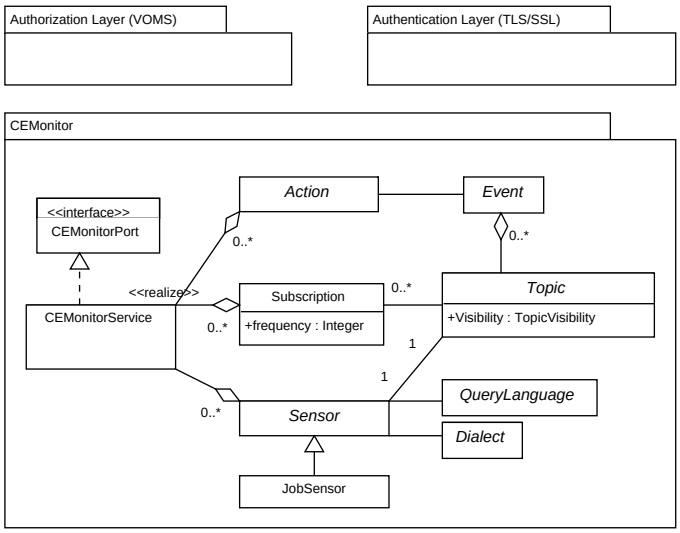

Figure 5: Internal structure of CEMonitor

the CREAM service where the job is being executed; `JOB_STATUS` is the current job status (in human-readable format); `TIMESTAMP` represents the time (in seconds since epoch) when the job status change happened; `WORKER_NODE` is the name of the execution host for the job. In this case, the job has not started execution yet, so the information on the worker node is reported as not available. Figure 7 shows an XML rendering of the same information.

```
[
  CREAM_JOB_ID = "CREAM986407854";
  CREAM_URL = "https://cream-02.pd.infn.it:8443/ce-cream/services/CREAM2";
  JOB_STATUS = "REGISTERED";
  TIMESTAMP = "1232444196000";
  WORKER_NODE = "N/A"
]
```

Figure 6: Job status change notification in classad Dialect

Table 4 lists all the operations supported by CEMonitor.

It must be stressed that CEMonitor is not strictly coupled with CREAM. It is instead a generic framework for information gathering and provisioning. For example in the context of the Open Science Grid (OSG) ReSS project is used to manage Grid resource information [18].

```
<status>
  <cream_job_id>CREAM986407854</cream_job_id>
  <cream_url>
    https://cream-02.pd.infn.it:8443/ce-cream/services/CREAM2
  </cream_url>
  <job_status>REGISTERED</job_status>
  <timestamp>1232444196000</timestamp>
  <worker_node>N/A</worker_node>
</status>
```

Figure 7: Job status change notification in XML Dialect

6 Putting the components together

In this section we summarize the submission to CREAM via the ICE enabled WMS with the UML Sequence Diagram shown in Fig. 8.

The relevant messages shown in the diagram are as follows:

1. ICE invokes the *getProxyReq* operation on the Delegation service. The request parameter is a string which represents the delegation ID which will be associated to the delegated credentials.

2. The delegation service replies with a CSR, which is a RFC3280 style proxy certificate request in PEM format with Base64 encoding [20].

3. ICE signs the CSR on behalf of the user which originally submitted the job. This is possible because ICE itself is using a user proxy certificate which has been delegated to the WMS. Then, ICE sends back: the ID of the already delegation session initiated on step 1, the RFC3280 style proxy certificate, signed by ICE on behalf of the user, in PEM format with Base64 encoding.

4. The Delegation service transfers the delegation ID/signed proxy to CREAM. Note that both CREAM and the delegation service execute on the same physical host, so they can communicate locally.

5. ICE requests the creation of a new lease, with a given lease ID. At the moment, ICE maintains a single lease for each user submitting jobs, so there are as many lease IDs as the number of unique users submitting to a specific CREAM CE. Note that the gLite WMS (and thus ICE) submits jobs on behalf of the user, using an X509 proxy certificate which has been delegated from the gLite UI (see Fig. 1) to the WMS.

Service Management Operations	
GetInfo	Gets information about the CEMonitor service, including the version and a brief description of the service, plus a list of available Topics and Actions.
Lease Management Operations	
Subscribe	Subscribes for notifications. The user specifies the Topic, a Query to be executed and a set of Actions to trigger when the Query succeeds. The notification rate can also be specified as parameter.
Update	Updates an existing Subscription: it is possible to modify the Topic, Query, triggered Actions and/or notification rate.
GetSubscriptionRef	Gets the list of all Subscription IDs and associated expiration times belonging to the caller.
GetSubscription	Gets detailed information on a set of Subscriptions given their unique IDs.
Unsubscribe	Removes an existing subscription. Events associated to that Subscription will no longer be notified.
PauseSubscription	Pauses the stream of notifications associated with a given Subscription ID.
ResumeSubscription	Resumes sending notifications associated with a previously paused Subscription.
GetTopics	Gets the list of Topics supported by CEMonitor.
GetTopicEvent	Gets the list of events associated with the specified Topic.

Table 4: CEMonitor interface operations

6. ICE is now ready to submit jobs to CREAM using the existing delegation ID and lease ID. The first step is to invoke the *JobRegister* operation on CREAM: this operation prepares the job for execution, by first creating some temporary files for internal use on the CE host.

7. The CREAM service registers the job, creates all the temporary files and returns a CREAM job ID which can be used from now on to refer to this job.

8. ICE invokes the *JobStart* operation, using the CREAM job ID as parameter, to request that the job is actually transferred to the LRMS, and to request that execution begins.

9. CREAM forwards the job to the LRMS; the job is added to the LRMS batch queue, and will eventually be executed.

10. ICE subscribes to CEMonitor to receive job status change notifications. This is done only if there are no active subscriptions on that specific CREAM CE; if so,

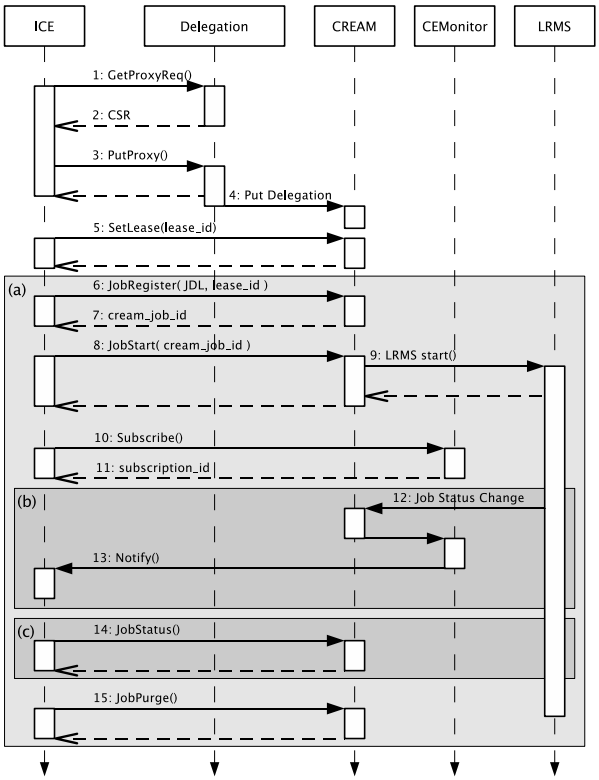

Figure 8: Overall job submission sequence diagram

there is no need to create a new subscription, as it is possible to use the existing one.

11. CEMonitor returns a Subscription ID, which can be used later on to renew, modify or cancel the subscription.

12. The LRMS, through the BLAH component (see Section 4), notifies CREAM about each job status change. CREAM in turn informs CEMonitor.

13. CEMonitor sends an appropriate notification to ICE; in order to reduce round-trip times, CEMonitor batches multiple related notifications which are sent together to subscribed clients.

14. ICE also periodically queries the job states directly to the CREAM service using the *JobStatus* operation.

15. When the job terminates, ICE invokes a *JobPurge* operation to remove all temporary files which have been created on the CE node.

We remark that it is necessary to perform a single delegation operation and to create a single lease for each user. So, after the first job has been submitted, all subsequent submissions for the same user will require the interactions shown in box (a) only. The interactions in box (b) are executed whenever CEMonitor has new job status changes to notify. We recall that, in order to improve efficiency, CEMonitor batches multiple status change events for the same user into a single notification which is sent to the clients each $1/f$ seconds, f being the user-defined notification frequency (in seconds). Finally, the interactions shown in box (c) are executed only when ICE does not receive status change notifications for some jobs for longer than a configurable threshold.

Finally, note that we omitted from Fig. 8 the operations required to renew the delegations when they are about to expire, and the operations required to renew the leases when they are about to expire. Delegation renewal involves exactly the same operations required for delegating credentials for the first time (operations 1 through 4 in the sequence diagram); lease renewal is performed by calling *SetLease* with an existing lease ID, as in operation 5 in the diagram.

7 Build, Installation and Usage

All the components of the gLite middleware (including CREAM and CEMonitor) are built using the ETICS Build and Test facility [8]. 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 [17] 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) [3]. 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.

8 Performance Considerations

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, considering the submission through the WMS. 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).

The layout of the testbed is shown in Fig. 9. 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 submit 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 the WMS UI to the WMS is not the bottle-

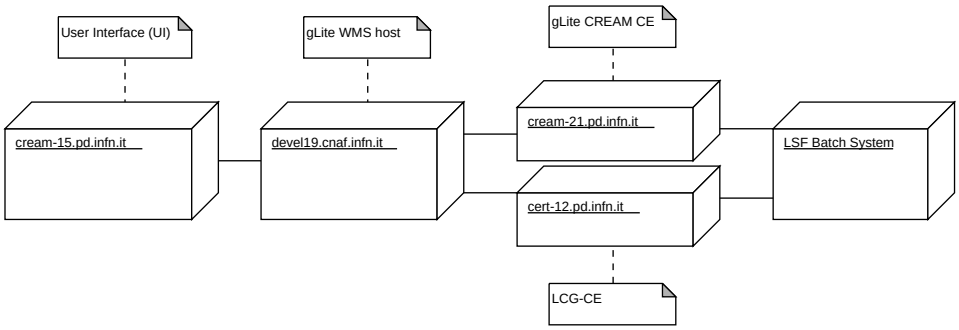

Figure 9: Layout of the testbed

neck in our tests, we execute the following steps:

1. We switch off the ICE or JC component of the WMS;
2. We submit 1000 jobs from the WMS UI;
3. 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 are 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:

- 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* (see Section 3.2). 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,

	Proxy Renewal	**Delegation**	**Submission Rate (jobs/sec)**	
			CREAM/ICE	LCG-CE/JC+CondorG+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 5: Test results; better submission rates are shown underlined

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 [21]; each test has been repeated two times, and the average of the measured submission rates is considered.

Table 5 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. As already described in section 3, CREAM exposes a delegation port-type to allow clients to securely delegate their credentials to the CE. The delegation operation (steps 1–4 from Fig. 8) 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 per-

forms 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.

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 [1] 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[2] 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 successfull 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.

9 Conclusions

In this paper we described CREAM and CEMonitor, two software components which are used in the gLite middleware to implement a job execution and management service. CREAM allows users to submit and manage computational jobs on a LRMS. CREAM provides additional features on the top of the underlying batch system, such as Grid-enabled user authentication and authorization, improved reliability, and integration with the rest of the gLite infrastructure. CEMonitor is a general-purpose event notification service, which can be coupled with CREAM to allow authorized users to receive notifications about job status changes without the need to explicitly poll the service.

CREAM and CEMonitor have been integrated into the gLite WMS using an additional component called ICE. ICE receives requests from the gLite WM, and handles all interactions with CREAM and CEMonitor. ICE takes care of delegating user credentials to the CREAM service, subscribing to CEMonitor for receiving job status change notifications, and actually submitting jobs to CREAM. ICE then monitors the jobs and

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

registers to the gLite LB service all status changes, such that Grid users know exactly the location and the status of their jobs.

CREAM and CEMonitor expose a Web Service interface, which allows easy interoperability with heterogeneous client applications. Recently, the Grid community is putting considerable effort in defining standard interfaces to Grid services. The reason for this interest is twofold: from one side, standard interfaces allow different middlewares to easily share resources and services. Furthermore, standards-compliant components improve the software development cycle by allowing developers to import software components from other middleware stacks. For these reasons, we implemented a prototype support for the BES and JSDL specifications in CREAM. However these specifications, in their current status, proved to be inappropriate for production use, as they only provide basic functionality. In particular, the JSDL specification is severely limited because it only allows users to describe simple (batch) jobs, while structured jobs such as collections of tasks with dependencies cannot be represented using the current JSDL standard. Furthermore, security considerations are outside the scope of the BES specification, which results in the possibility for different services to claim standard-compliance without being interoperable due to the use of mutually incompatible security settings. To address these problems, the Grid community is currently defining production-quality extensions of the BES and JSDL interfaces, which will eventually be implemented in CREAM and will replace the legacy interface.

CREAM and CEMonitor are being actively developed. In future releases we plan to improve the scalability and fault tolerance of these services by implementing appropriate clustering/failover mechanisms. Clustered configuration allows multiple service instances to balance their load, and can also be used to tolerate failures. However, as CREAM and CEMonitor are both stateful services, special care must be taken such that each instance share the same internal status while avoiding single points of failure. We are also investigating how some ideas from the *cloud computing* paradigm could be integrated into CREAM. In particular, we are considering the possibility of dynamically adjusting the size (number of hosts) of the underlying LRMS to allow the system to automatically scale whenever needed. This could be done, for example, by implementing a LRMS based on Amazon's EC2 service, such that the batch system pool could be dynamically increased by instantiating new virtual hosts.

Acknowledgments

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

References

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

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

[3] YAIM Home Page. http://yaim.info/.

[4] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, Á. Frohner, K. Lőentey, and F. Spataro. From gridmap-file to VOMS: managing authorization in a Grid environment. *Future Generation Computer Systems*, 21(4):549–558, 2005.

[5] Paolo Andreetto et al. The gLite Workload Management System. *Journal of Physics, Conference Series*, 119(6):062007 (10pp), 2008.

[6] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Adreas Savva. Job Submission Description Language (JSDL) Specification, Version 1.0, November 2005. OGF Specification GFD-R.056, http://www.gridforum.org/documents/GFD.56.pdf.

[7] Apache Software Foundation. Axis SOAP Container. http://ws.apache.org/axis/.

[8] Marc-Elian Bégin, Guillermo Diez-Andino Sancho, Alberto Di Meglio, Enrico Ferro, Elisabetta Ronchieri, Matteo Selmi, and Marian Zurek. Build, configuration, integration and testing tools for large software projects: Etics. In Nicolas Guelfi and Didier Buchs, editors, *RISE*, volume 4401 of *Lecture Notes in Computer Science*, pages 81–97. Springer, 2006.

[9] Stephen Burke, Simone Campana, Elisa Lanciotti, Patricia Mendez Lorenzo, Vincenzo Miccio, Christopher Nater, Roberto Santinelli, and Andrea Sciabà. *gLite 3.1 User Guide–Manuals Series*, January7 2009. Version 1.2, Document identifier CERN-LCG-GDEIS-722398. Available online at https://edms.cern.ch/document/722398/1.2.

[10] CEMonitor home page. http://grid.pd.infn.it/cemon.

[11] Paul DuBois. *MySQL*. Addison-Wesley Professional, 2008.

[12] M. Ellert, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J.L. Nielsen, M. Niinimäki, O. Smirnova, and A. Wäänänen. Advanced resource connector middleware for lightweight computational grids. *Future Generation Computer Systems*, 23(2):219–240, 2007.

[13] Dietmar W. Erwin. UNICORE–a grid computing environment. *Concurrency and Computation: Practice and Experience*, 14:1395–1410, 2002.

[14] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service Version 1.0, August 2007. OGF Specification GFD.108, `http://www.ogf.org/documents/GFD.108.pdf`.

[15] Ian Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *IFIP International Conference on Network and Parallel Computing*, pages 2–13, 2005.

[16] Ian Foster et al. Modeling Stateful Resources with Web Services. White paper, March 5 2004. Version 1.1, Available online at `http://www.ibm.com/developerworks/library/ws-resource/ws-modelingresources.pdf`.

[17] Eric Foster-Johnson. *Red Hat RPM Guide*. Red Hat, 1st edition, March1 2003.

[18] Gabriele Garzoglio, Tanya Levshina, Parag Mhashilkar, and Steve Timm. ReSS: A Resource Selection Service for the Open Science Grid. In Simon C. Lin and Eric Yen, editors, *Grid Computing, International Symposium on Grid Computing (ISGC 2007)*, pages 89–98. Springer, 2009.

[19] D Groep, O Koeroo, and G Venekamp. gLExec: gluing grid computing to the Unix world. *Journal of Physics: Conference Series*, 119(6):062032 (11pp), 2008.

[20] R. Housley, W. Polk, W. Ford, and D. Solo. RFC3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. `http://www.ietf.org/rfc/rfc3280.txt`, April 2002.

[21] Raj Jain. *The Art of Computer System Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley, 1991.

[22] D. Kouřil et al. Distributed tracking, storage, and re-use of job state information on the grid. In *Proceedings of CHEP'04*, Interlaken, Switzerland, September27–October1 2004.

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

[24] Site authorisation and enforcement services: LCAS and LCMAPS. `http://www.nikhef.nl/grid/lcaslcmaps/`.

[25] E. Molinari et al. A local Batch System Abstraction Layer for Global Use. In *Proc. XV International Conference on Computing in High Energy and Nuclear Physics (CHEP'06)*, Mumbay, India, February13–17 2006.

[26] Rajesh Raman. *Matchmaking Frameworks for Distributed Resource Management*. PhD thesis, University of Wisconsin-Madison, 2001.

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

[28] Sun Microsystems, Inc. *Java^{TM} Platform Enterprise Edition, v5.0, API Specifications*, 2007.

[29] Douglas Thain, Todd Tannenbaum, and Miron Livny. Distributed computing in practice: the Condor experience. *Concurrency–Practice and Experience*, 17(2-4):323–356, 2005.

[30] Apache Software Foundation. Jakarta Tomcat Servlet Container. `http://tomcat.apache.org/`.

[31] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. RFC3820: Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. `http://www.ietf.org/rfc/rfc3820.txt`, June 2004.

[32] R. van Engelen. *gSOAP 2.7.11 User Guide*, October 2 2008.