Job submission and management through web services: the experience with the CREAM service

C Aiftimiei[1,5], P Andreetto[1], S Bertocco[1], D Cesini[3], M Corvo[2], S Dalla Fina[1], S Da Ronco[1], D Dongiovanni[3], A Dorigo[1], A Gianelle[1], C Grandi[4], M Marzolla[1], M Mazzucato[1], V Miccio[2], A Sciaba'[2], M Sgaravatto[1], M Verlato[1] and L Zangrando[1]

[1] INFN Sezione di Padova, Via Marzolo 8, 35131 Padova, Italy
[2] CERN, BAT. 28-1-019, 1211 Geneve, Switzerland
[3] INFN CNAF, viale Berti Pichat 6/2, 40127 Bologna, Italy
[4] INFN Sezione di Bologna, viale Berti Pichat 6/2, 40127 Bologna, Italy

Abstract. Modern Grid middlewares are built around components providing basic functionality, such as data storage, authentication, security, job management, resource monitoring and reservation. In this paper we describe the Computing Resource Execution and Management (CREAM) service. CREAM provides a Web service-based job execution and management capability for Grid systems; in particular, it is being used within the gLite middleware. CREAM exposes a Web service interface allowing conforming clients to submit and manage computational jobs to a Local Resource Management System. We developed a special component, called ICE (Interface to CREAM Environment) to integrate CREAM in gLite. ICE transfer job submissions and cancellations from the Workload Management System, allowing users to manage CREAM jobs from the gLite User Interface. This paper describes some recent studies aimed at assessing the performance and the reliability of CREAM and ICE; those tests have been performed as part of the acceptance tests for integration of CREAM and ICE in gLite. We also discuss recent work towards enhancing CREAM with a BES and JSDL compliant interface.

1. Introduction

Many Grid middlewares are large software artifacts which provide a set of basic functionalities, each one implemented by a separate component. Such functionalities include (but are not limited to) data storage, authentication and authorization, resource monitoring, and job management. In particular, the job management component is used to submit, cancel, and monitor jobs for execution on a suitable computational resource, also called Computing Element. A Computing Element (CE) has a complex structure: it represents the interface to a usually large farm of computing hosts managed by a Local Resource Management System, such as LSF or PBS. Moreover, a CE should also provide additional features than those of the underlying batch system, such as Grid-enabled user authentication and authorization, accounting, fault tolerance and improved performance and reliability.

Computing Resource Execution and Management is a system designed for efficiently manage a CE in a Grid environment. The goal of Computing Resource Execution and Management

[5] On leave from NIPNE-HH, Romania

(CREAM) is to offer a simple, robust and lightweight service for job operations. CREAM 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 users can use any language with a Web Service framework to generate their own client interfaces. CREAM is a Java application running as an extension of a Java-Axis servlet inside the Tomcat application server [1].

CREAM is being developed within the gLite Grid infrastructure [2]. A special component (called Interface to Cream Environment) has been created to integrate CREAM into the gLite Workload Management System (WMS), so that gLite users can submit jobs to a CREAM server in a transparent way. Note, however, that CREAM is a mostly self-contained service, with few dependencies on the gLite software components. Users can thus install CREAM in stand-alone mode, and interact directly with it through custom clients, or using the provided C++-based command line tools.

In this paper we describe the CREAM architecture and highlight its features. We describe its interface, and show how it has been integrated with the gLite infrastructure. Finally, we show the results of performance tests which have been done in order to assess CREAM throughput, compared to the throughput of the other CEs used in gLite.

2. CREAM overview

CREAM main functionality is job submission: users can submit jobs, described via a classad-based Job Description Language (JDL) expression [3], to CREAM based CEs. CREAM JDL is the same notation used to describe job characteristics and requirements in the gLite WMS. The JDL is a high-level, user-oriented language based on Condor classified advertisements (classads) [4] for describing jobs to be submitted to the CREAM CE service. 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 also produce a set of output data files. The set of input files is called InputSandBox (ISB), while the set of files produced by the application is called 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:

```
[
  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:

- `file1` and `file3` will be copied from the client (User Interface (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 job completes its execution.

Other typical job management operations (job cancellation, job status with different level of verbosity and filtering, job listing, job purging) are supported. Moreover users are allowed to suspend and then restart jobs submitted to CREAM based CEs, provided that the underlying Local Resource Management System (LRMS) supports this feature. Direct job submission and management to CREAM can be done with a set of command-line tools (written in C++); of course, users can produce their own client application which uses CREAM Web Service Description Language (WSDL) interface. Job submission is also possible through the gLite Workload Management System, through the ICE component (discussed later).

For what concerns security, authentication (considering a GSI based framework) is properly supported in all operations. Authorization on the CREAM service is implemented, supporting both Virtual Organization (VO) based policies and policies specified on the single Grid users.

3. CREAM architecture

Fig. 1 shows the typical deployment of a CREAM server. CREAM runs as a Java-Axis servlet [5] on the Tomcat application server [1]. CREAM interacts with CEMON [6], which provides asynchronous job status change notification service, through a CREAM backend which implements the Java Java Naming and Directory Interface (JNDI) interface. Requests to CREAM and CEMON traverse a pipeline of additional components which take care of authorization issues. CREAM submits requests to the LRMS through Batch-system Local ASCII Helper (BLAH), an abstraction layer for the underlying LRMS. BLAH, in turn, interacts with the client-side LRMS environment, which might consists of a set of command line tools which interact with a server-side LRMS

The CREAM service is available through a Web Service interface. The use of Web Services is one of the key features of CREAM. CREAM is intended to offer job management facilities to the widest range possible of consumers. This includes not only other components of the same middleware, but also single users and other heterogeneous services. Thus, we need a mechanism that let potential users to be as much free as possible in using their own tools and languages to interface to CREAM. The Web Services technology offers all the interoperability characteristics that are needed to fulfill the above requirements. From the implementation point of view, CREAM is a Java application which uses Apache Tomcat as application server; CREAM executes inside an Axis container, and exposes a SOAP interface.

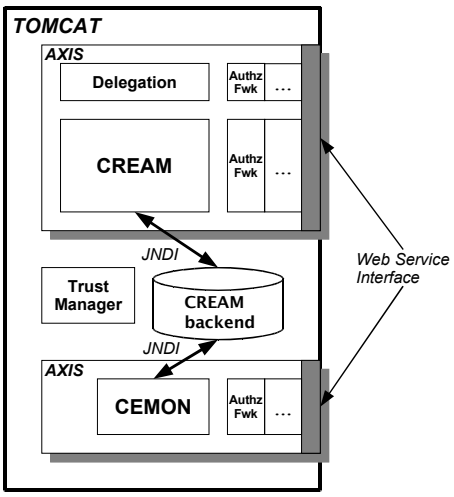

Figure 1. Typical deployment of a CREAM server

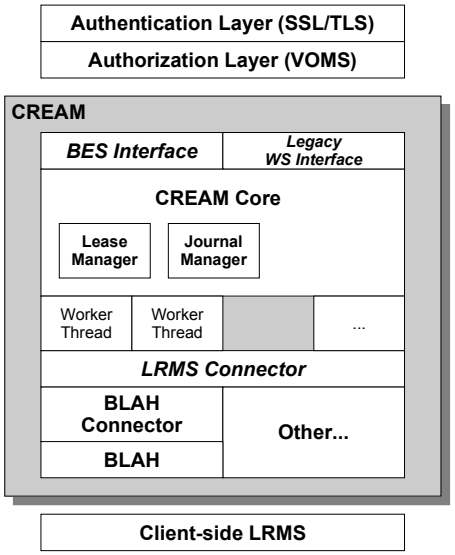

Figure 2. CREAM layered architecture

Fig. 2 shows a layered view of the CREAM service. Names in italic denote interfaces, the other ones denoting actual components. The figure shows the internal structure of CREAM, which we now discuss in detail.

3.1. CREAM interfaces

CREAM exposes two different interfaces: the *legacy* one, and the new BES-compliant interface (the latter still under development). Basic Execution Service (BES) is a recent specification [7] for a standard interface to execution services provided by different Grid systems. The aim of Basic Execution Service (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 three Web Services *port-types* as follows:

BES-Factory This port-type allows clients to create, monitor, and control sets of jobs, and to monitor BES attributes, such as the number of jobs currently being instantiated. This port-type is intended for use by ordinary clients.

BES-Activity This port-type allows clients to create, monitor, and control individual activities. This port-type is intended for use by ordinary clients.

BES-Management This port-type allows clients to monitor the details of and control the BES itself. This port-type is intended for use by system administrators.

BES uses the Job Submission Description Language (JSDL) [8] 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 expiration while a job is running).

3.2. CREAM backend

The CREAM backend is a permanent storage where CREAM saves the data related to the jobs it is managing. The CREAM backend is implemented as a custom Java-based persistent storage mechanism which implements the Java Naming and Directory Interface [9] interface. CREAM also creates a directory for each user that successfully registered a job. The directory, which is located on the filesystem of the CREAM server, contains all informations about the job, such as its description in form of a job JDL, the certificate used by the user to submit it, etc.

3.3. Journal Manager

The Journal Manager (JM) is a pool of threads of the CREAM main process. User job commands (job submission requests, job cancellations, etc.) are enqueued into the JM, which stores them on persistent storage to preserve them in case of system failure. The JM then serves these requests, interacting with the underlying LRMS through BLAH. The JM is used to parallelize job submission: multiple job management commands are simultaneously forwarded to the LRMS to improve the overall throughput. Commands submitted by the same user are executed sequentially in the order they were received by CREAM.

3.4. Lease Manager

When job submissions arrive through the gLiteWMS, it is essential that all jobs submitted to CREAM eventually reach a terminal state (and thus get eventually purged from the CREAM server). The gLite WMS has been augmented with an additional component, Interface to Cream Environment (ICE), which is responsible for directly interacting with CREAM. ICE and CREAM use a lease-based approach to ensure that all jobs get 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 an appropriate `JobLease` CREAM method. ICE is responsible for periodically renewing all leases which are about to expire. Should a job lease expire before the actual termination of a job, the Lease Manager thread will purge it and free all the CE resources used by that job.

3.5. LRMS connectors

As already introduced, CREAM can be seen as an abstraction layer on top of a 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*. A LRMS connector is an interface for a generic batch system. Currently, CREAM supports all the batch systems supported by BLAH [10] through a specific instance of LRMS connector called *BLAH connector module*: at the time of writing BLAH supports LSF, PBS, and Condor. Of course, it is also possible to create other, ad-hoc connectors to interact with other types of batch systems.

3.6. Security

The Grid is a large collaboration and resource sharing environment. Users and services cross the boundaries of their respective organizations and then 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 VOs and at the level of single resource owners. Both these aspects must be taken into account. Moreover, as we will see in the next sections, Grid services have to face the problem of dealing with the delegation of certificates and the mapping of Grid credential 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 is an implementation of the J2EE security specifications.

Authentication in CREAM is based on a Public Key Infrastructure (PKI). Each user (and Grid service) willing to access CREAM is required to present an X.509 format certificate [11]. 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 user main data 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 certificate with the CA public key. This ensures that the certificate was released 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*. Proxy certificates are an extension of X.509 certificate, using RFC3820 proxy-certificates [12]. 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 portType* of the service. CREAM creates a public-private key pair and use it to generate a Certificate Sign Request (CSR). This a certificate that has to be signed by the user with her private key. The signed certificate is 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 as 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 polices 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 at the administrator choice 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 his 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 denied access.

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 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). Waiting for the development 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, a sudo-style program which allows executing the user's job with local credential 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) [13]. glexec also uses the Local Centre Authorization Service (LCAS) [14] 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. WMS integration

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 [15].

CREAM functionality can also be used by the gLite WMS Grid component [16]. This means that jobs submitted to the gLite WMS can be forwarded for their 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 (i.e. the trigger of a possible resubmission if a Grid failure is detected).

Fig. 3 shows a schematic view of CREAM integration with the gLite WMS. gLite users submit jobs through the gLite 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. Of course, in both cases the users must have a valid VOMS proxy credential. Other clients can also access the BES interface to CREAM.

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 a separate component called CEMON (see Fig. 1). CEMON [6] is a general-purpose event notification framework. CREAM notifies the CEMON component about job state changes by using the shared, persistent CREAM backend. ICE subscribes to CEMON 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)

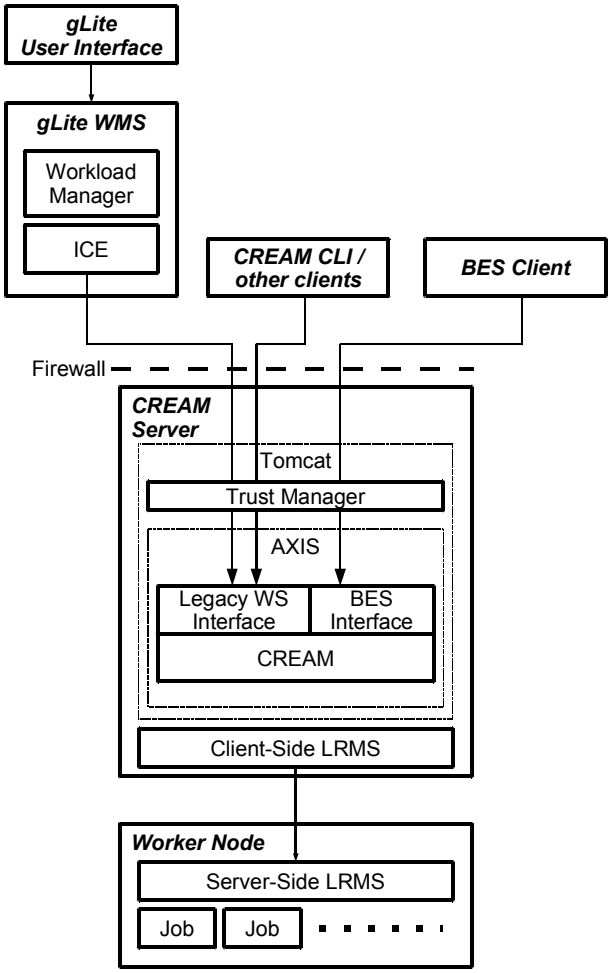

Figure 3. Integration of CREAM within the gLite WMS infrastructure; the BES interface is currently used by stand-alone BES clients.

even if the CEMON service becomes unavailable. Job status change informations are sent to the Logging and Bookkeeping (LB) [17] service, a distributed job tracking service.

5. Performance tests

We evaluated the performance of the CREAM service in term of throughput (number of submitted jobs/s), comparing CREAM with the CEs currently used in the gLite infrastructure. The CEs which have been tested are CREAM, the LCG-CE and the gLite-CE. We measured the throughput for the submission of 1000 identical jobs to an idle CE, using an otherwise identical infrastructure. All 1000 jobs were inserted into the input queue of the WMS component responsible for submitting jobs to the CE–that is, the input queue of ICE for the CREAM CE, and the JobController/LogMonitor (JC+LM) input queue for the gLite-CE and LGC-CE. When the input queue was full, ICE (or JC+LM) were switched on to start the submission to the CE. The following JDL was submitted in all tests:

```
Executable = "test.sh";
StdOutput = "std.out";;
InputSandbox = {"gsiftp://grid005.pd.infn.it/Preview/test.sh"}
OutputSandbox = "out.out";
OutputSandboxDestURI = {"gsiftp://grid005.pd.infn.it/Preview/Output/std.out"};
ShallowRetryCount = 0;
RetryCount = 0;
```

with `test.sh` being:

```
#!/bin/sh
echo "I am running on ‘hostname‘"
echo "I am running as ‘whoami‘"
sleep 600
```

Those jobs had the ISB (the executable) downloaded from a GridFTP server, and had the output (the stdout, where the hostname where the execution took place and where the local user are reported) uploaded on a GridFTP server as well. Each job did a sleep for 10 minutes. Deep and shallow resubmission were disabled.

The following measures were taken:

(i) The number of successful jobs, that is, the number of jobs which successfully completed their execution (we recorded the reported failure reasons for all failed jobs); this is an Higher is Better (HB) metric.

(ii) Throughput to the CE. This is a HB metric, which denotes the number of jobs/minute which are received by the CE;

(iii) Throughput to the LRMS. This is a HB metric, which denotes the number of jobs/minute which are received by the LRMS.

In the case of submissions to CREAM using ICE, the test was repeated considering different values for the number of submitting threads in ICE. The results are given in Table 1. Note that since in the JC+Condor+LM scenario it is not straightforward to distinguish between submission to the CE vs submission to the LRMS, it was not possible to measure the submission rate to the CE as done for the ICE case.

Table 1. Performance results

	Successful jobs	Throughput for CE submission (jobs/60s)	Throughput for LRMS submission (jobs/60s)
ICE/CREAM, 5 threads	999/1000	37.36	37.29
ICE/CREAM, 10 threads	994/1000	37.56	37.48
ICE/CREAM, 15 threads	995/1000	37.34	37.22
ICE/CREAM, 20 threads	994/1000	36.45	36.39
ICE/CREAM, 25 threads	994/1000	38.83	38.73
LGC-CE	1000/1000	N/A	13.52
gLite-CE	940/1000	N/A	2.38

We should note that the JC+Condor+LM to gLite-CE tests revealed a problem in Condor so that it only could handle 100 jobs, even if the maximum value was correctly set to 1000 in Condor's configuration file. At the time the tests were performed, a fix for this bug was not available yet. This partly explains the lower submission rates for the LGC-CE and gLite-CE.

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 features 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 informations, including installation instructions, interface specification and usage manual for CREAM can be found at the Web page [18].

7. Acknowledgments

EGEE is a project funded by the European Union under contract INFSO-RI-508833. The OMII-Europe project is funded by the European Union under contract number INFSO-RI-031844. We also acknowledge the national funding agencies participating in EGEE and/or OMII for their support of this work.

References

[1] Apache software foundation. jakarta tomcat servlet container `http://tomcat.apache.org/`
[2] Burke S, Campana S, Peris A D, Donno F, Lorenzo P M, Santinelli R and Sciabà A 2007 *gLite 3 user guide–Manuals Series* Version 1.1, Document identifier CERN-LCG-GDEIS-722398. Available online at `https://edms.cern.ch/document/722398/1.1`
[3] 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`
[4] Raman R 2001 *Matchmaking Frameworks for Distributed Resource Management* Ph.D. thesis
[5] Apache software foundation. axis SOAP container `http://ws.apache.org/axis/`
[6] CEMon home page `http://grid.pd.infn.it/cemon`
[7] Foster I *et al.* 2007 OGSA basic execution service, version 1.0 available at `http://www.ogf.org/documents/GFD.108.pdf`
[8] Anjomshoaa A *et al.* 2007 Job submission description language (JSDL) specification, version 1.0 available at `http://www.ogf.org/documents/GFD.56.pdf`
[9] Java naming and directory interface (JNDI) `http://java.sun.com/products/jndi/`
[10] Molinari E *et al.* 2006 *Proc. XV International Conference on Computing in High Energy and Nuclear Physics (CHEP'06)* (Mumbay, India)
[11] Housley R *et al.* RFC3280: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile `http://www.ietf.org/rfc/rfc3280.txt`
[12] Tuecke S Rfc3820: Internet x.509 public key infrastructure (pki) proxy certificate profile `http://www.ietf.org/rfc/rfc3820.txt`
[13] `http://www.nikhef.nl/grid/lcaslcmaps/lcmaps.shtml`
[14] Site authorisation and enforcement services: LCAS and LCMAPS `http://www.nikhef.nl/grid/lcaslcmaps/`
[15] van Engelen R gSOAP 2.7.6 user guide 29 Dec. 2005
[16] Andreetto P *et al.* 2004 *Proceedings of CHEP'04* (Interlaken, Switzerland)
[17] Kouřil D *et al.* 2004 *Proceedings of CHEP'04* (Interlaken, Switzerland)
[18] CREAM home page `http://grid.pd.infn.it/cream`