CREAM: A SIMPLE, GRID-ACCESSIBLE, JOB MANAGEMENT SYSTEM FOR LOCAL COMPUTATIONAL RESOURCES

P. Andreetto, S. A. Borgia, A. Dorigo, A. Gianelle, M. Marzolla, M. Mordacchini, M. Sgaravatto, L. Zangrando, INFN Sezione di Padova, Via Marzolo 8, I-35131 Padova, Italy
F. Dvořák, D. Kouřil, A. Křenek, L. Matyska, M. Mulač, J. Pospíšil, M. Ruda, Z. Salvet, J. Sitera, J. Škrabal, M. Voců, CESNET z.s.p.o., Zikova 4, 160 00 Praha 6, Czech Republic
G. Avellino, S. Beco, A. Cavallini, A. Maraschini, F. Pacini, A. Parrini, C. Scarcella, M. Sottilaro, A. Terracina, DATAMAT S.p.A., via Laurentina 760, I-00143 Roma, Italy
S. Monforte, M. Pappalardo, INFN Sezione di Catania, via S. Sofia 64, I-95123 Catania, Italy
S. Andreozzi, M. Cecchi, V. Ciaschini, T. Ferrari, F. Giacomini, R. Lops, E. Ronchieri, V. Venturi, INFN CNAF, viale Berti Pichat 6/2, I-40127 Bologna, Italy
G. Fiorentino, V. Martelli, M. Mezzadri, E. Molinari, F. Prelz, D. Rebatto, INFN Sezione di Milano, via Celoria 16, I-20133 Milano, Italy
A. Guarise, G. Patania, R. Piro, A. Werbrouck, INFN Torino, Via P. Giuria 1, I-10125 Torino, Italy

Abstract

Efficient and robust system for accessing computational resources and managing job operations is a key component of any Grid framework designed to support large distributed computing environment. Computing Resource Execution and Management (CREAM) is a simple, minimal system designed to provide efficient processing of a large number of requests for computation on managed resources. Requests are accepted from distributed clients via a Web Service based interface. The CREAM architecture is designed to be a robust, scalable and fault tolerant service of a Grid middleware. In this paper we describe the CREAM architecture and the provided functionality. We also discuss how CREAM is integrated within the EGEE gLite middleware in general, and with the gLite Workload Management System in particular.

INTRODUCTION

One of the most important functionality of Grid systems is managing job operations: users can submit, cancel, and monitor jobs submitted for execution on a Computing Element (CE). A CE has a complex structure: it represents the interface towards 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 apart from those of the underlying batch system, such as Grid-enabled user authentication and authorization, accounting, fault tolerance and improved performance.

CREAM is a system designed for efficiently manage a CE in a Grid environment. The goal of 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. CREAM is a Java application running as an extension of a Java-Axis servlet inside the Tomcat application server [2].

In this paper we describe the CREAM architecture and highlight its features.

CREAM FUNCTIONALITY

CREAM main functionality is job submission: users can submit jobs, described via a classad-based Job Description Language (JDL) expression, to CREAM based CEs. CREAM JDL is the same language used to describe job characteristics and requirements in the gLite Workload Management System (WMS). CREAM supports the execution of batch and parallel (MPI) jobs; the support of bulk jobs (i.e. parametric jobs and collections of independent jobs) is ongoing. CREAM also allows to transfer the Input Sandbox (ISB) to the executing node; the ISB is a set of files needed for the execution of the job, that must be transferred from the client node and/or from Grid storage servers. The other typical job management operations (job cancellation, job status with different level of verbosity and filtering, job listing, job purging) are supported as well. Moreover users are allowed to suspend and then restart jobs submitted to CREAM based CEs.

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 VO based policies and policies specified on the single Grid users.

CREAM ARCHITECTURE

Fig. 1 shows CREAM internal components. The CREAM application runs as a Java-Axis servlet on the Tomcat application server. CREAM interacts with CEMON, which provides asynchronous job status change notification service, through a CREAM backend which

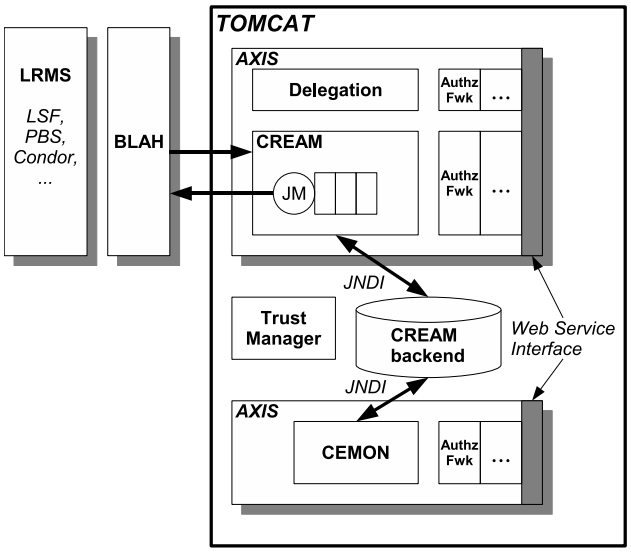

Figure 1: CREAM internal architecture

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 contains a separate thread (the Journal Manager) which submits requests to the Local Resource Management System (LRMS) through an additional component (BLAH) acting as an abstraction layer for the underlying LRMS.

The CREAM services are 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 [2] as application server; CREAM executes inside an Axis [1] container, and exposes a SOAP interface.

Cream Backend

The CREAM backend is a permanent memory space where CREAM stores 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 [5]. CREAM also creates a directory for each user that successfully registered a job. The directory 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.

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. BLAH [11] is an abstraction layer which provides a uniform interface to different batch systems. 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 initially issued, to maintain the causal relationships between commands.

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 Virtual Organization (VO)s 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 [9]. 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 which 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 allow 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 DN) 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) [6]. glexec also uses the Local Centre Authorization Service (LCAS) [7] 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.

WMS INTEGRATION

CREAM services can be accessed directly by the user. A set of command line utilities which can be used to manage jobs by directly invoking CREAM methods have in fact been developed. These command line tools are written in C++ using the gSOAP library [13].

CREAM functionality can also be used by the gLite WMS Grid component [8]. 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 being done by developing a separate module, called Interface to Cream Environment (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 (see Fig. 2) 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).

The state of a job can be obtained in two different ways. The first one is by subscribing to a job status change notification service implemented by a separate component called

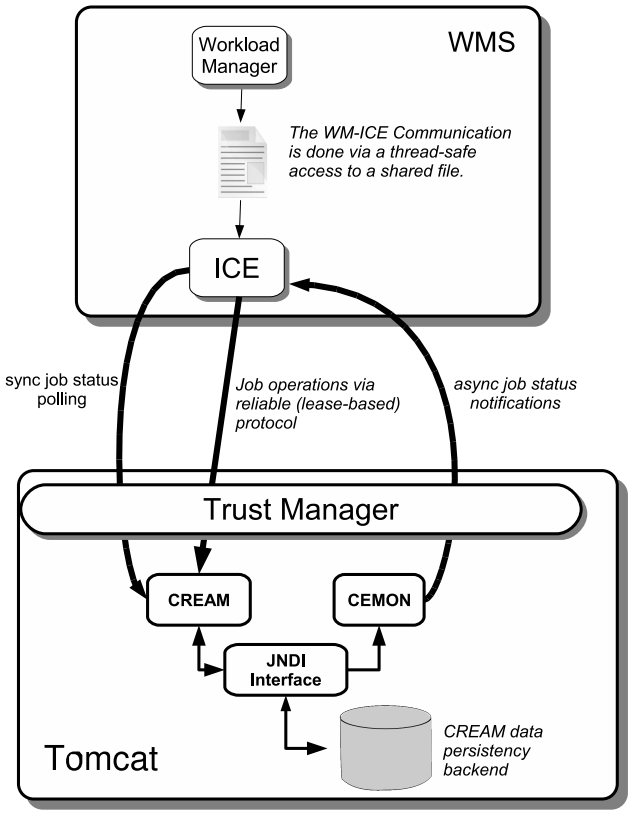

Figure 2: CREAM-WMS integration

CEMON. CEMON [3] 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) even if the CEMON service becomes unavailable. Job status change informations are sent to the Logging and Bookkeeping (LB) [10] service, a distributed job tracking service.

In order to guarantee that the set of jobs managed by ICE is consistent with those on the CREAM servers, ICE implements a lease-based protocol to signal its interest in some jobs. The protocol works as follows: each CREAM job has a *lease time*: if the lease time expires and the job is not terminated yet, CREAM stops and purges the job. ICE is responsible to periodically renew the lease for all jobs it is interested to monitor.

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.

More detailed informations, including installation instructions, interface specification and usage manual for CREAM can be found at the Web page [4].

ACKNOWLEDGMENTS

EGEE is a project funded by the European Union under contract INFSO-RI-508833. We also acknowledge the national funding agencies participating in EGEE for their support of this work.

REFERENCES

[1] Apache software foundation. axis soap container. `http://ws.apache.org/axis/`.

[2] Apache tomcat home page. http://tomcat.apache.org/.

[3] CEMON home page. `http://grid.pd.infn.it/cemon/field.php`.

[4] CREAM home page. `http://grid.pd.infn.it/cream/field.php`.

[5] Java naming and directory interface (JNDI). `http://java.sun.com/products/jndi/`.

[6] Local credential mapping service (LCMAPS) home page. `http://www.nikhef.nl/grid/lcaslcmaps/lcmaps.shtml`.

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

[8] P. Andreetto et al. Practical approaches to grid workload and resource management in the EGEE project. In *Proceedings of CHEP'04*, Interlaken, Switzerland, 27 Sept.–1 Oct. 2004.

[9] R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, Apr. 2002. Available at `http://www.ietf.org/rfc/rfc3280.txt`.

[10] D. Kouřil et al. Distributed tracking, storage, and re-use of job state information on the grid. In *Proceedings of CHEP'04*, Interlaken, Switzerland, 27 Sept.–1 Oct. 2004.

[11] E. Molinari et al. A local batch system abstraction layer for global use. In *Proc. CHEP'06*, Mumbai, India, 13–17 Feb. 2006. To Appear.

[12] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 public key infrastructure (PKI) proxy certificate profile, June 2004. Available at `http://www.ietf.org/rfc/rfc3820.txt`.

[13] R. van Engelen. *gSOAP 2.7.6 User Guide*, 29 Dec. 2005.