New developments in the CREAM Computing Element

Paolo Andreetto\textsuperscript{a}, Sara Bertocco\textsuperscript{a}, Fabio Capannini\textsuperscript{b}, Marco Cecchi\textsuperscript{b}, Alvise Dorigo\textsuperscript{a}, Eric Frizziero\textsuperscript{a}, Alessio Gianelle\textsuperscript{a}, Massimo Mezzadri\textsuperscript{c}, Salvatore Monforte\textsuperscript{d}, Francesco Prelz\textsuperscript{c} David Rebatto\textsuperscript{c}, Massimo Sgaravatto\textsuperscript{a} and Luigi Zangrando\textsuperscript{a}

\textsuperscript{a} INFN Padova, Via Marzolo 8, I-35131 Padova, Italy
\textsuperscript{b} INFN CNAF, Viale Berti Pichat 6/2, I-40127 Bologna, Italy
\textsuperscript{c} INFN Milano, Via Celoria 16, I-20133 Milano, Italy
\textsuperscript{d} INFN Catania, Via Via Santa Sofia, 64, I-95123 Catania, Italy

E-mail: paolo.andreetto@pd.infn.it, sara.bertocco@pd.infn.it, fabio.capannini@cnaf.infn.it, marco.ceccib@cnaf.infn.it, alvise.dorigo@pd.infn.it, eric.frizziero@pd.infn.it, alessio.gianelle@pd.infn.it, massimo.mezzadri@mi.infn.it, salvatore.monforte@ct.infn.it, francesco.prelz@mi.infn.it, david.rebatto@mi.infn.it, massimo.sgaravatto@pd.infn.it, luigi.zangrando@pd.infn.it

Abstract. The EU-funded project EMI aims at providing a unified, standardized, easy to install software for distributed computing infrastructures. CREAM is one of the middleware products part of the EMI middleware distribution: it implements a Grid job management service which allows the submission, management and monitoring of computational jobs to local resource management systems. In this paper we discuss about some new features being implemented in the CREAM Computing Element. The implementation of the EMI Execution Service (EMI-ES) specification (an agreement in the EMI consortium on interfaces and protocols to be used in order to enable computational job submission and management required across technologies) is one of the new functions being implemented. New developments are also focusing in the High Availability (HA) area, to improve performance, scalability, availability and fault tolerance.

1. Introduction

CREAM (Computing Resource Execution and Management) \cite{1} provides a job execution and management service for Grids. It is the gLite implementation of a Computing Element (CE): the interface to a usually large farm of computational resources managed by a Local Resource Management System (LRMS). It also implements a few features that aren’t ordinarily provided by the underlying batch systems, such as Grid enabled user authentication and authorization, accounting, fault tolerance and improved performance and reliability.

CREAM exposes a Web Service interface allowing conforming clients to submit, manage and monitor computational jobs.

The main functionality of CREAM is job management. Users submit jobs described via a Job Description Language (JDL) \cite{2} expression, and CREAM executes them on the underlying...
batch system. The JDL is a high level notation based on Condor classified advertisements (classads) [3] 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 [4].

A simplified view of the architecture of CREAM is shown in fig. 1.

CREAM is written in Java, and runs as an extension of a Java-Axis servlet inside the Apache Tomcat application server.

Requests to CREAM traverse a pipeline of components which take care of security issues. The TrustManager is the component responsible for carrying out PKI-based authentication. It is an implementation of the J2EE security specifications.

The aim of the Authorization Handler is instead to check whether an authenticated user has the rights to access services and resources to perform certain operations. Authorization in CREAM can be managed through gJAF (Grid Java Authorization Framework), which provides a way to invoke a chain of policy engines and get a decision result about the authorization of a user. Authorization can also be implemented via ARGUS [5], a system meant to provide consistent authorization decisions for distributed services.

CREAM uses an external relational database to store information about jobs and user delegation proxies. This improves fault tolerance as it guarantees that this information is preserved across restarts of CREAM. Moreover, the use of a database improves responsiveness of the service while performing queries which are needed by the normal CREAM operations. The database server can be installed on a dedicated host, or can share the same machine as CREAM.

Input and output files are instead stored in the local file-system (job sandboxes) and can be accessed remotely via a gridFTP server.

CREAM submits requests to the LRMS through BLAH [6], an abstraction layer providing a unified interface to 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. At the time of writing, BLAH supports LSF, PBS/Torque, Grid Engine and Condor. CREAM relies on BLAH also for receiving job status change notifications from the LRMS.
Users can interact directly with the CREAM service by means of a set of command line utilities which manage jobs by directly invoking CREAM operations.

CREAM can also be used through higher level job management services: in particular the gLite WMS component and Condor-G [7] both support the submission to CREAM CEs.

The evolution and maintenance of CREAM occur in the context of the European Middleware Initiative (EMI) project [8]. EMI brings together the major middleware providers in Europe (in particular ARC, gLite, UNICORE and dCache) to deliver a consolidated set of middleware components for deployment in several Grid infrastructures at both the national and the international level.

EMI major releases are delivered once per year. The first one (codename Kebnekaise) was released on May 2011. The second major release (Matterhorn) is being released at the time of writing, while the Monte Bianco release, the third and last one one, will be released at the end of February 2013.

In this paper we discuss about some new developments that are either introduced with the second major release of EMI or are currently being implemented.

2. EMI Execution Service

Job management services developed in the context of different middleware stacks implement job description and job management in different, incompatible ways, even if the provided core functionality is largely the same.

This was done mainly because of the unavailability of consolidated and usable standards in the area. Actually some standard mechanisms for job description and standard interfaces for job submission and job management do exist (namely BES [9] and JSDL [10]), but they are not really suited for production use because they lack significant capabilities, e.g. for data staging.

One of goal of the EMI project is to address this issue, to enable interoperability in the job management area across the ARC, gLite and UNICORE relevant services. Since the current efforts in the PGI OGF working group [11], which follow an extremely slow top-down approach, don’t match the EMI schedule, a specific task force composed by ARC, gLite and UNICORE representatives was formed to address this issue. The task force output was the specification of an EMI Execution Service [12] (or “EMI-ES” for short) targeted to Computing Elements. Higher level services, such as workload managers, brokering services, or workflow systems, were instead considered out of scope.

This specification covers the interfaces to create and manage activities (i.e. jobs), and the Activity Description Language (EMI-ADL), i.e. an XML dialect for describing the activity characteristics and its resource requirements.

Support for data staging capabilities is part of the specification. For what concerns the staging of input data, both client data push (input data staged in the execution service by the client) and server data pull (input data retrieved from one or remote servers) are foreseen. The same push and pull modes are foreseen for output data as well. Data staging in EMI-ES is shown in fig. 2

The specification also describes the support for delegation of client credentials needed for data staging: for the time being only the delegation of X.509 credentials was considered, while SAML [13] has been reserved for the future. Activity and resource related information is also covered by this specification.

The EMI-ES interface was split in 5 port types for modularity:

- Creation
- ActivityManagement
Figure 2. Data staging in EMI-ES.

- Delegation
- ActivityInfo
- ResourceInfo

The EMI-ES architecture consists of two main modules. The first module is the Activity-Factory, responsible to create activities and manage resource (CE) information. It implements the Create, ResourceInfo and the Delegation port-types. The second module is the Activity-Manager, responsible to manage activities and activity related information. It implements the ActivityManagement, ActivityInfo and Delegation port-types.

In the CREAM-CE released in the EMI-2 version, a first support for EMI-ES is provided: besides installing CREAM with the legacy interface, also the EMI-ES interface can therefore be optionally deployed.

The following functionality is provided via the EMI-ES interface:
- job submission of single or multiple activities;
- job management operations: cancel, suspend, resume, list, wipe of activities;
- status and info activity related operations;
- proxy delegation operations: creation, renewal, and information about delegations;
• support for client data pull and server data push;
• support for multiple sources for input files, and multiple targets for output files.

A EMI-ES Command Line Interface (CLI) is also part of the EMI second major release: this
allows end users to use the functionality provided by any EMI-ES Computing Element (not only
the CREAM EMI-ES implementation).

The ResourceInfo Port-type, used to get information about the CREAM CE resource in Glue
2.0 format, isn’t implemented in the EMI-2 release yet. It will be released in a subsequent EMI-2
update.

3. CREAM High Availability
A request that many Grid infrastructures are posing on the CREAM CE is the ability to be
continuously available for serving user requests, even in case of planned and unplanned outages.
The request translates into transparently accessing a set of individual CEs through one CE alias
name.

Partial fulfillment of this requirement is already available in the current CREAM
implementation: it is already possible to define a DNS alias to refer to a set of CREAM head
nodes. After the initial contact from outside clients to the CREAM-CE alias name for job
submission, all further actions on that job are based on the jobid which contains the physical
hostname of the CREAM-CE to which the job was submitted. This allows to switch the DNS
alias in order to distribute load. There are various techniques to change an alias entry in the
DNS, e.g. using self-made scripts or referring to commercial network techniques: the choice
depends strongly on the way the network is set up and managed.

However this implementation has some problems: in particular it doesn’t fully work when
the interaction with CREAM is done by the gLite-WMS and the Condor-G systems.

A new architecture to implement high availability and to improve the overall scalability
and performance had therefore to be designed and it is now being implemented. The goal was
to maintain the CREAM architecture as simple as possible and limit new developments.

The idea is to rely on a cluster of commodity computers, seen as a single service, the same
way as many popular services on the Internet. In this architecture, we call CREAM node a single
CREAM instance running on its dedicated (physical or virtual) machine, while a collection of
such nodes is referred as CREAM cluster. The overall load should be spread among all the
available nodes of the CREAM cluster. Moreover it should be possible to add/remove/replace
nodes from the cluster on the fly, without shutting down or interrupting the operations of the
overall system.

A high level view of the envisaged CREAM clustered architecture is shown in fig. 3.

A WEB server (e.g. apache) acts as gateway for incoming requests of authenticated users.
These requests are then delivered to the load balancer which redirects each of them to a proper
CREAM node. The policies and the algorithm to be used to choose this CREAM node have
still to be fully assessed; possible choices are e.g. a simple Round Robin, or a Weight based
algorithm.

Each CREAM node is first of all responsible to apply the authorization rules on the incoming
request. Authorized requests are then processed, exactly as in the non-clustered architecture.

Job management requests will still be stored in the persistent command queue (implemented
through a relational database) as in the non-clustered architecture. But in this case this
command queue will be logically centralized and accessed concurrently by all the CREAM nodes
of the cluster. So, every CREAM instance can process requests independently of whomever was
queuing them. Given that by design CREAM is a stateless Web Service which treats each
request as independent, no explicit session replication is required.
Since the database is a potential Single Point Of Failure (SPOF), it must itself be clustered. Implementation solutions already exist (e.g. MySQL Cluster, which partitions DB tables across nodes and assuring high availability).

The job sandboxes, which contain input and output data files accessed/produced during the job’s life cycles, will be stored in a filesystem shared among all CREAM nodes of the cluster (implemented e.g. through NFS) and accessible remotely via gridFTP.

The implementation of CREAM High Availability as described above is foreseen with the third EMI major release.

4. Other new developments
Another new development that is planned to be finalized by the end of the EMI project is the integration with the EMI Common AuthenTication Library (CaNL) [14]. This common library, supposed to be adopted by all EMI middleware services, has been designed for several purposes: to favor interoperability, to reduce the code and ease its maintenance, to facilitate code security audits. This library consists of a set of objects and functions dealing with user authentication using X.509 TLS, while authentication using different mechanisms (SAML, Kerberos, LDAP, OpenID, etc.) is out of scope.
The CaNL library, whose implementation is being finalized, is available in different languages. The Java implementation will be integrated on the CREAM server side, while the C/C++ version will be used for the CREAM UI.

Another new development foreseen in the third year of the EMI project is the support of the Simple Linux Utility for Resource Management (SLURM) batch system, as requested by several user communities. Such support must be mainly implemented in the BLAH component, but some efforts will be needed also for the dynamic batch system providers and for the accounting sensors.

LSF, Torque/PBS, Grid Engine and partly Condor are instead already supported.

5. Conclusions
In this paper we discussed some new features in the CREAM Computing Element, just released or being implemented.

More detailed information can be found on the CREAM web page [15].

Acknowledgments
The research results of the EMI project are co-funded by the European Commission under the FP7 Collaborative Projects Grant Agreement Nr. 261611.

References
[1] C. Aftimiei et al, Design and Implementation of the gLite CREAM Job Management Service, Future Generation Computer Systems, Volume 26, Issue 4, April 2010, pp. 654-667, doi: 10.1016/j.future.2009.12.006.
[2] CREAM Job Description Language Attributes Specification, available online at https://wiki.italiangrid.it/twiki/bin/view/CREAM/JdlGuide.
[3] R. Raman, Matchmaking Frameworks for Distributed Resource Management, Ph.D. thesis, University of Wisconsin-Madison, 2001.
[4] P. Andreetto et al., Evolution of Grid meta-scheduling towards the EMI era, in Proc. of CHEP’10, 18th International Conference on Computing in High Energy and Nuclear Physics, 18-22 October 2010, Taipei (Taiwan)
[5] ARGUS Website, https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework
[6] E. Molinari et al, A local batch system abstraction layer for global use, Proc. of XV International Conference on Computing in High Energy and Nuclear Physics (CHEP’06), Feb 13–17 2006, Mumbai, India.
[7] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: A Computation Management Agent for Mulled-Institutional Grids Proc. of of HPDC 2001.
[8] European Middleware Initiative (EMI) project WebSite, http://www.eu-emi.eu/home
[9] I. Foster et al, OGSA Basic Execution Service, Version 1.0, available online at http://www.ogf.org/documents/GFD.108.pdf
[10] A. Anjomshoaa et al., Job Submission Description Language (JSDL) Specification, Version 1.0, available online at http://www.gridforum.org/documents/GFD.56.pdf
[11] OGF Production Grid Infrastructure WG (PGI-WG) WebSite, https://forge.ogf.org/sf/projects/pgi-wg
[12] B. Schuller (editor), B. Konya, O. Smirnova, A. Konstantinov, M. Skou Andersen, M. Riedel, S. Memon, S. Memon, L. Zangrando, M. Sgaravatto, E. Frizziero, EMI Execution Service Specification, available online at https://twiki.cern.ch/twiki/bin/view/EMI/EmiExecutionService
[13] Security Assertion Markup Language (SAML) v2.0 Standard set, available online at https://www.oasis-open.org/standards/#samlv2.0
[14] EMI Common Authentication Library WebSite, https://twiki.cern.ch/twiki/bin/view/EMI/EMIraIT4SecurityCommonAuthNLib
[15] CREAM Website, https://wiki.italiangrid.it/CREAM