Future Generation Computer Systems 26 (2010) 654–667

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Design and implementation of the gLite CREAM job management service

Cristina Aiftimiei[a,1], Paolo Andreetto[a], Sara Bertocco[a], Simone Dalla Fina[a], Alvise Dorigo[a], Eric Frizziero[a], Alessio Gianelle[a], Moreno Marzolla[b,*], Mirco Mazzucato[a], Massimo Sgaravatto[a], Sergio Traldi[a], Luigi Zangrando[a]

[a] *Istituto Nazionale di Fisica Nucleare (INFN), via Marzolo 8, I-35131 Padova, Italy*

[b] *Dipartimento di Scienze dell'Informazione, Università di Bologna, Mura A. Zamboni 7, I-40127 Bologna, Italy*

A R T I C L E   I N F O

Article history:
Received 4 May 2009
Received in revised form
10 December 2009
Accepted 14 December 2009
Available online 21 December 2009

Keywords:
Web services
gLite middleware
Grid computing
Grid job management service
Notification service

A B S T R A C T

Job execution and management is one of the most important functionalities provided by every modern Grid systems. 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 Grids, while CEMonitor is a general purpose asynchronous event notification framework. Both components expose a Web Service interface allowing conforming clients to submit, manage and monitor computational jobs to a Local Resource Management System.

1. Introduction

Grid middleware distributions are often large software artifacts, which include a set of components 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 [1].

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 [2] middleware distribution and currently in production use within the EGEE Grid infrastructure [3]. Users can install CREAM in stand-alone

* Corresponding author. Tel.: +39 051 2094847; fax: +39 051 2094510.
E-mail addresses: cristina.aiftimiei@pd.infn.it (C. Aiftimiei), paolo.andreetto@pd.infn.it (P. Andreetto), sara.bertocco@pd.infn.it (S. Bertocco), simone.dallafina@pd.infn.it (S. Dalla Fina), alvise.dorigo@pd.infn.it (A. Dorigo), eric.frizziero@pd.infn.it (E. Frizziero), alessio.gianelle@pd.infn.it (A. Gianelle), marzolla@cs.unibo.it (M. Marzolla), mirco.mazzucato@pd.infn.it (M. Mazzucato), massimo.sgaravatto@pd.infn.it (M. Sgaravatto), sergio.traldi@pd.infn.it (S. Traldi), luigi.zangrando@pd.infn.it (L. Zangrando).
[1] On leave from NIPNE-HH, Romania.

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 developed. ICE receives job submission and cancellation requests coming from a gLite WMS, and forwards these requests to CREAM. ICE then handles the entire lifetime of a job, including registering each status change to the gLite Logging and Bookkeeping (LB) service [4].

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, each service must take into account the specific features of the middleware it belongs to.

The UNICORE (Uniform Interface to Computing Resources) [5] 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) [6] is a Grid middleware developed by the NorduGrid collaboration. ARC is based on the Globus Toolkit,[2] 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 [7]. GRAM4 includes a set of WSRF-compliant Web Services [8] 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.

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 LCG-CE related to performance and security issues. These issues and other requirements behind the development of CREAM will be discussed in Section 3.1.

1.2. Organization of this paper

This paper is organized as follows. In Section 2 we give a high level overview on the job management chain in the gLite middleware. Then, in Section 3 we restrict our attention on the CREAM and CEMonitor services: we illustrate the requirements defined in the gLite design document for the Computing Element, and give a high level description of CREAM and CEMonitor. Internal details on CREAM are given in Section 4, and details on CEMonitor are given in Section 5. The interactions with CREAM and CEMonitor which are necessary to handle the whole job submission sequence are then explained in Section 6. Section 7 describes how CREAM and CEMonitor are built and deployed in the gLite production infrastructure. Section 8 contains performance considerations, and we discuss conclusions and future works 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 [2,9] 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 JobController+LogMonitor+CondorG and LCG-CE components are 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 [10]. In Fig. 1 we have emphasized the paths from the WMS UI to CREAM (top) and to the legacy LCG-CE (bottom).

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 [11]. The CREAM CLI provides a set of commands to invoke the Web Services operations exposed by CREAM (the list of available operations is given in Section 4).

On the other hand, the gLite WMS UI allows the user to submit and monitor jobs through the gLite Workload Management System (WMS) [12]. 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 to 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 cannot 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 WMSs can be associated with 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 submissions, 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

[2] Globus and Globus Toolkit are trademarks of the University of Chicago.