TOWARD THE CREAM CLUSTERING

Introduction

Today CREAM can be considered (Computing Resource And Management) a mature service for job management. Facing the challenge of supporting an expanding community of users with new requirements, as part of the European Middleware Initiative (EMI), such service needs to be consolidated and evolved.
One of the main objectives described in its evolution plan, is the need to meet the High Availability (HA) criteria. In particular, CREAM, like several popular Internet services (e.g. Google, Amazon), must rely on large clusters of commodity computers for providing several features, including high performance, scalability, availability and fault tolerance. From a user's point of view the main benefit provided by this enhancement is the guaranteed access to her jobs and related resources (i.e. job sandbox) during planned and unplanned outages. For these reasons, we are focusing on providing CREAM with the ability to be continuously available to serve- user requests independently of critical conditions which might arise.

Objectives

Enhancing CREAM with High Availability is the main objective; this implies the need to modify its architecture to support clustering technology. There are several levels of complexity in implementing a cluster of services and the best solution depends on what we really need. Often sophisticated clustered systems are supported by dedicated private networks just for the execution of heartbeats, status and control activities. Moreover, specific hardware devices (with comprehensive internal hardware redundancy) are utilized to guarantee high performance and efficiency under heavy stress conditions. Of course expensive servers, RAID disk arrays and fiber channels are welcome if available, but are out of scope. Paradoxically, adding more components to an overall system design can undermine efforts to achieve high availability. That is because complex systems inherently have more potential failure points and are more difficult to implement correctly.
So, our intent is to keep the CREAM architecture simple by limiting the development of new components and taking advantage from the existing ones. Moreover all software components should be carefully replicated in order to avoid potential bottlenecks and SPOFs (Single Point Of Failure).
In the context of CREAM clustering, we call “CREAM node” a separate CREAM instance running on its dedicated (virtual) machine, while a collection of such nodes is referred to as CREAM cluster. In order to maintain the same level of serviceability/availability (QoS) in increased load conditions, the CREAM nodes should share their work load. In the event of a (un)scheduled downtime, the involved nodes should be dropped from the cluster without shutting down the overall system; these nodes should also be replaced by backup nodes with a hot deploy. Moreover the cluster should be load balanced and the whole system complexity should of course be hidden to the user, who is not interested in distinguishing a single CREAM service from the clustered one.
As long term objective, we plan to provide the support of the cluster partitioning for hosting a subset of these CREAM nodes on a physically different location. This would allow for continued operation even in the case of a disaster occurring at one of the CREAM sites.

The current CREAM architecture

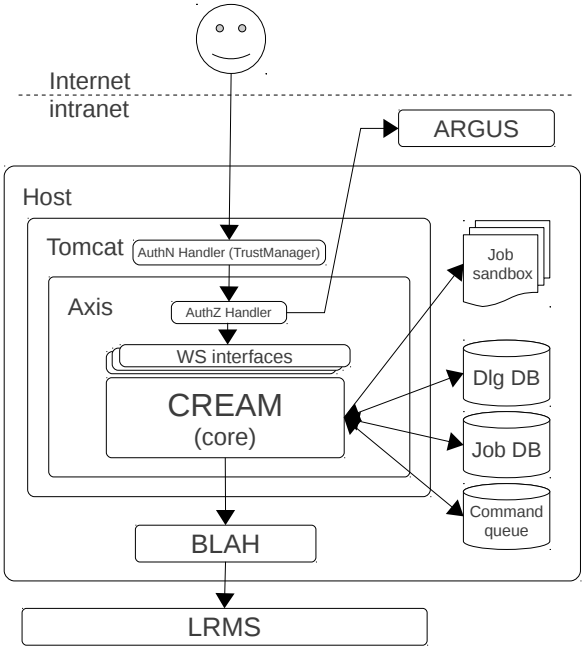

figure 1: the current CREAM architecture

The relevant software components involved in the current CREAM-CE architecture are schematically shown in figure 1. The CREAM business logic has been fully implemented in Java and its functionalities are all exposed by a legacy Web Service interface which guarantees a high interoperability level. Moreover the CREAM service is executed inside the Axis container deployed in the Apache Tomcat application server. The user requests travel along a pipeline of additional components which, for example, take care of authentication (TrustManager) and authorization issues (ARGUS) or act as an abstraction layer for interacting with the underlying LRMS (BLAH). Static and dynamic information about jobs and user delegation proxies are persistently stored in the local database (MySQL) while all files accessed or produced during the job's life cycle are stored in the local file system (job sandboxes) and accessible remotely via a gridFTP client.

The CREAM clustered architecture

The implementation of a clustered CREAM service entails the merging and/or logical centralization of information coming from all CREAM instances in the cluster; this affects several services that CREAM uses, such as databases, sandboxes, BLAH, etc (figure 2). In order to avoid SPOF in these services, these should also undergo some form of replication.
The figure 2 illustrates the high level CREAM clustered architecture which is based on a horizontal topology. In particular the logical centralization of information and all software components composing the cluster is highlighted. For simplicity, replications are explicitly omitted from the figure. The WEB server (e.g. Apache) acts as gateway for incoming requests of authenticated users. These requests are delivered to the load balancer which redirects them to the proper CREAM nodes taking decisions based on the selected scheduling algorithm (e.g. Round Robin, Weight based, etc). Moreover the load balancer can even provide fault tolerance capability, if appropriately configured. In turn the CREAM node applies the authorization rules and, if allowed,

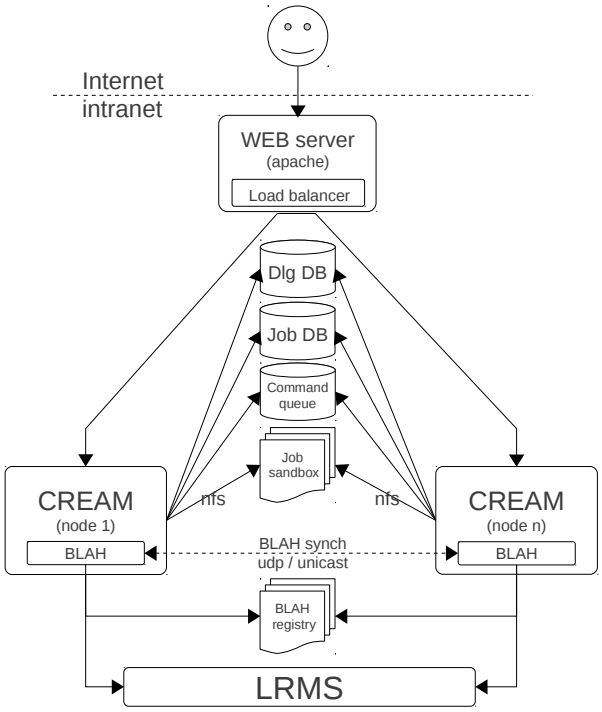

figure 2: the high level CREAM clustered architecture

processes the requests exactly like in the current architecture. Please note that the use of the WEB server is just a possible approach for implementing the load balancing. Other more or less sophisticated solutions could be implemented by the site administrator. For instance, a DNS server may be instructed to forward user requests to the CREAM nodes on a scheduling algorithm based on the simplest Round Robin (RR).

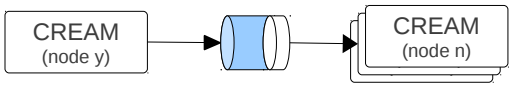

figure 3: the shared command queue

Requests for job management will still be stored in the persistent command queue which, contrary to the current architecture, will be logically centralized and accessed concurrently by the CREAM nodes. So, every CREAM instance can process requests independently of whoever was queuing them (figure 3). This capability has already been implemented and no further development is needed. Furthermore, given that by design CREAM is a stateless Web Service which treats each request independently, no explicit session replication is required.
Static and dynamic job information, including delegation proxies will be stored in the (logically) centralized database (MySQL) while the job sandboxes which contain all input and output data files accessed and/or produced during the job's life cycles, will be stored in the shared file system (e.g. NFS, GPFS) and still be accessible remotely via gridFTP (figure 4).

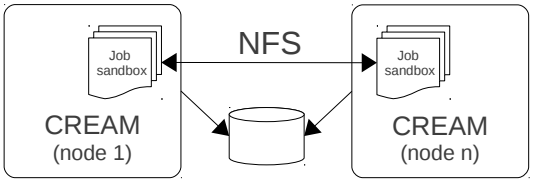

figure 4: the shared job information

Since the database is a potential SPOF, it must itself be clustered. The best approach relies on a concept such as a MySQL Cluster, which involves splitting the software components that allow the data access (SQL processes) from the data itself so that SQL processes and data will be hosted in different machines. Both processes and data must be replicated, in particular data redundancy could be obtained at the physical (e.g. RAID) and/or logical (data replicas) level. The number of logical replicas denotes the number of copies of the same data item within the database. Operations executed against a data item, are replicated over all its replicas and the transaction will terminate only when that operation is successfully completed against the last replica. This implies the existence of primary (first utilized) and secondary (the true copies) replicas.

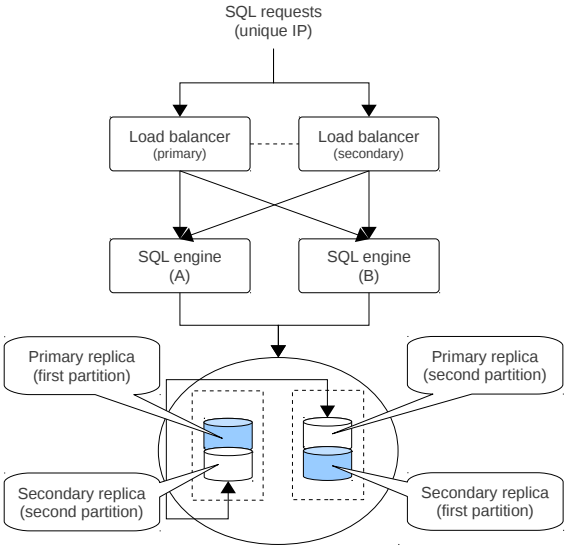

figure 5: logical structure of a clustered database

Figure 5 shows the scheme of the ideal solution which requires the distribution of the database on at least, two separate systems each one containing its own primary replicas and the secondary ones of the other system. So, in the event of damage or whatever event compromising the activity of one system, the overall database is still consistent and the data access guaranteed. In particular the access will be always available by exploiting two SQL engines, through a unique IP address handled by a load balancer which will distribute the SQL requests. To make the scheme SPOF free, the load balancer should be replicated.
Finally, to implement the illustrated system in a realistic fashion, at least two dual CPU machines equipped with 4 GB of RAM each, are needed.

By analogy with the logical centralization of CREAM information, all relevant data handled by BLAH must be centralized too (figure 2). On large batch systems with a high throughput of jobs it has been observed that status requests can easily overload the LRMS daemon. To mitigate the problem BLAH is provided with its own cache for all the batch system related information on submitted jobs. This cache, called "job registry", is implemented as a flat file with some in-memory indexes to speed up access. A daemon called BUpdater running on the CREAM node itself, periodically updates the local registry, refreshing old information with (optimized) queries to the batch system. Unfortunately, the centralization cannot be implemented by adopting the same approach as CREAM, since, in order to be a lightweight component, BLAH limits the adoption of third-party software components (e.g. SQL databases) in favor of custom solutions. A possible approach which solves the issue of the external dependencies and even guarantees good performances is discussed below and illustrated in figure 6. In a HA cluster environment, each BUpdater daemon running on different CREAM nodes would have to keep its registry synchronized with all other nodes. In order to avoid proliferation of the queries, a new feature has been added to the daemon. Whenever it refreshes the local registry from the LRMS, it also sends the updated information either to a multicast address or to a list of unicast addresses, where the other BUpdater daemons are listening. In turn, all BUpdaters update their registry without having to ask the batch system. Moreover this synchronization mechanism guarantees that new BLAH instances are updated whenever they are created. This completes the CREAM clustering design.

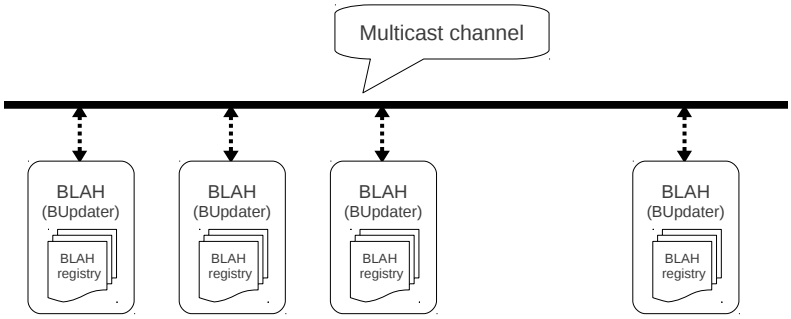

figure 6: logical structure of the clustered BLAH