JAliEn - A new interface between the AliEn jobs and the central services

A.G. Grigoras[1], C. Grigoras[1], M.M. Pedreira[1], P. Saiz[1], S. Schreiner[2]

[1] CERN, 1211 Geneva, Switzerland
[2] Technische Universität Darmstadt, Hochschulstraße 10, 64289 Darmstadt, Germany

E-mail: alina.grabriela.grigoras@cern.ch, costin.grigoras@cern.ch, miguel.martinez.pedreira@cern.ch, pablo.saiz@cern.ch, steffen.schreiner@cased.de

Abstract. Since the ALICE experiment began data taking in early 2010, the amount of end user jobs on the AliEn Grid has increased significantly. Presently 1/3 of the 40K CPU cores available to ALICE are occupied by jobs submitted by about 400 distinct users, individually or in organized analysis trains. The overall stability of the AliEn middleware has been excellent throughout the 3 years of running, but the massive amount of end-user analysis and its specific requirements and load has revealed few components which can be improved. One of them is the interface between users and central AliEn services (catalogue, job submission system) which we are currently re-implementing in Java. The interface provides persistent connection with enhanced data and job submission authenticity. In this paper we will describe the architecture of the new interface, the ROOT binding which enables the use of a single interface in addition to the standard UNIX-like access shell and the new security-related features.

1. Introduction

The ALICE [1] experiment uses AliEn (ALIce ENvironment) [2] middleware to leverage heterogeneous computing resources and provide users with a unified view of the system. It allows users to execute jobs wherever there are free resources matching the job requirements and transparently access existing data or upload job results to any of the 70 disk and tape storage elements.

The system currently has a set of databases at its core, holding the details of all experiment files, all jobs to be executed and the scheduled data transfers. All interactions with the databases go through a set of central services that enforce authentication, authorization and access to the stored objects and also to the physical files stored on the distributed storage infrastructure.

Current AliEn is assembled from about 200 packages written in Perl, C and C++, with two distinct channels of communicating to the central services: via web services or through xrootd-based communication channels. Similarly there are two distinct command line user interfaces, a full Perl implementation for site administration and a mix of bash scripting and C/C++ code for normal users.

In order to reduce the complexity of this system and streamline the operation and maintenance of this environment we have started to rewrite the core functionality in Java, profiting from the built-in serialization, easy to use fully TLS [5]-based communication, object caching and other powerful features. The result of these efforts is the jAliEn[3] Grid middleware project. Below are the details of the implementation so far and the plans for the future development needed in order to fully replace the current communication channels.

2. System architecture

The entire system is now a single Java package that maps each logical entity from the current AliEn implementation to a corresponding serializable object encapsulating the details (LFN, GUID, PFN, SE, JDL, Job, principals, quotas etc). To retrieve such objects an API based again on serializable objects extending a base Request class is offered to the users, similar to a Java Remote Method Invocation [4]. These requests encapsulate requester identity which is checked on the authoritative entities. Furthermore, some of the requests can implement a Cacheable interface, allowing them to be cached in memory of the various components and returned immediately to the requester instead of being forwarded for resolving to the authoritative entities. This is only done for long-lasting objects like SE details or GUID replica locations.

2.1. Overall layout

Services layout follows closely the current AliEn services layout:

- JCentral: a central component that handles the direct access to the core databases
- JSite: a connection multiplexing and caching service running on trusted site machines
- JBox: end-user (or job) service handling client-side authentication and upstream connections

In Figure 1 below the current interaction model in AliEn is presented in comparison with the Java implementation.

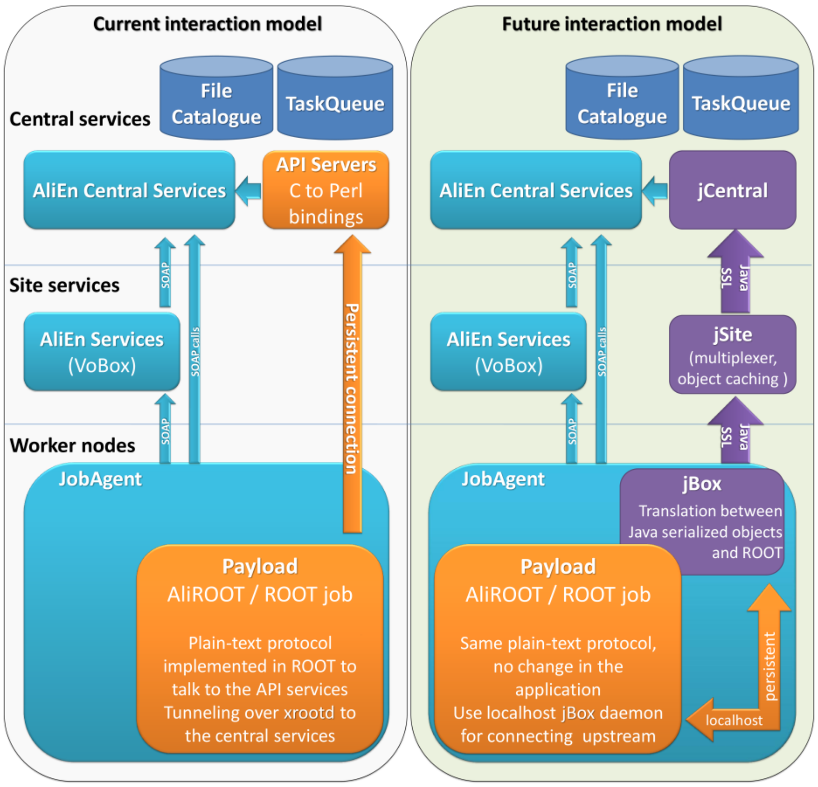
Current interaction model
Central services
File Catalogue
TaskQueue
AliEn Central Services
API Servers C to Perl bindings
Site services
SOAP
SOAP calls
AliEn Services (VoBox)
Persistent connection
Worker nodes
JobAgent
Payload AliROOT / ROOT job
Plain-text protocol implemented in ROOT to talk to the API services Tunneling over xrootd to the central services
Future interaction model
File Catalogue
TaskQueue
AliEn Central Services
jCentral
Java SSL
jSite (multiplexer, object caching)
AliEn Services (VoBox)
SOAP
SOAP calls
JobAgent
jBox Translation between Java serialized objects and ROOT
Payload AliROOT / ROOT job
Same plain-text protocol, no change in the application Use localhost jBox daemon for connecting upstream
persistent
localhost

Figure 1. Current and future interaction models in AliEn

Communications of jAliEn components are exclusively based on mutually authenticated and encrypted connections based on the TLS [5] protocol while using the Bouncy Castle Crypto API [6]. The authentication is established using the Grid user's X.509 [7] certificate and X.509 host certificates on service side.

2.2. JCentral

This component is responsible for handling the direct database connections and being the authoritative source to resolve any API requests. In practice the service listens on a single TCP port and expects TLS [5] connections to be established.

The AliEn principle is established based on the DN of the certificate used to connect to the service. This identity is then tagged to any Request object received on the ObjectStream. Any JCentral instance is able to answer any received Request, allowing the service to be horizontally scaled on as many central machines as needed. This is different from the current AliEn services where different capabilities are served by different services listening on different TCP ports, also with different security constraints and server technologies.

The aim was to have a minimalistic API and so far there are 25 distinct Request object types that cover all aspects of interacting with the central databases: getting logical and physical file details, performing list and find operations, manipulating catalogue entities, managing jobs and getting trace information and so on. As such it is up to the client to split a more complex request in a suite of such atomic operations in order to perform the entire command. For example a command to submit a job based on the specifications in a logical file (JDL) present in the catalogue would be translated in series of requests to the central services to locate the physical copies of the file, to get the access tokens to them, to actually retrieve the content, parse it on the client side and interpret the requirements, do the same for other files referenced from the JDL and finally prepare the entry that is to be injected in the central task queue. This delegates an important part of the processing to the entity that requests the service, while currently all these operations are executed centrally by the AliEn services and it frequently happens for them to accumulate a backlog of requests to process since the processing depends on external entities.

2.3. JSite

In order to avoid establishing persistent connections from each entity in the environment (which currently would be more than 50000 concurrent jobs and some hundreds of users, but at least a 4x increase has to be foreseen) this entity is used to multiplex requests on a single upstream connection. The same TLS [5] mechanism is used to decorate the requests, with the difference that the upstream connection will trust that the identity was verified by this layer and will not overwrite it any more. This implies that the services will run in a trusted environment, similarly to how the current AliEn site services are running, on site-local machines dedicated to the experiment.

In addition to forwarding the Request objects upstream and dispatching the replies to the corresponding channel this service is also able to hold Cacheable-implementing objects in memory and returns the cached reply directly to the client, as long as the entry is still valid. The time to live is set by the authoritative service (JCentral). All services that see this object will cache it in memory, so that if a similar request is generated within the lifetime of the reply it can be resolved directly in the memory of the client.

While this layer is designed to be cascaded indefinitely, in practice a single layer of services distributed on the site VOBox-es is enough to handle the site-generated message traffic (since even the largest sites currently run less than 10000 jobs to one VOBox). This leaves the option to scale the services either horizontally (several service instances on large sites) or vertically (for example with regional concentrators).

2.4. JBox
This service runs as a daemon on the end-user machine or in the environment of each job. It handles the authentication upstream based on the full X.509 certificate. This is again different from the current procedure in AliEn, requiring Grid users to periodically create a password-less X.509 proxy certificate [8], which conceptually represent temporary certificates as powerful as the main certificate itself. It will also require one-time X.509 certificates to be handed to each job for authentication as well, while currently jobs authenticate with a shared secret which is a unique token associated in the central task queue to every job ID.

JBox acts like ssh-agent, running as a daemon on the user machine and creating a temporary file in which connection details of the current instance are stored (local TCP port, shared secret). The file is protected with regular permissions that only allow the same user to read its content (0400). JBox will only establish upstream connections (to JSite or directly to JCentral) when it receives a command to execute. Idle connections are automatically closed after a timeout in order to reduce the load on the upstream services. A configurable timeout can be set on the authentication itself, to force the user to resupply the key password before being able to continue interacting with the environment. This improves the security over the current system by not having any unprotected certificate or proxy certificate stored persistently on the client's file system at any moment.

JBox listens on a local-machine TCP port only (the first available one in the 10100 and 10200 range). Along with the TCP port the process ID of the current daemon is written in the same temporary file. Subsequent attempts to start a service for the same account verify if the specified service is still running and exit immediately in that case as a protection against wrong commands.

This component is also responsible for parsing the commands received from the user prompt and executing the intended operations on the client side. For this it makes use of dynamic class loading in Java to instantiate a class with a name derived from the invoked command and pass it the arguments received from the command line. This way it is very easy to add new commands either in the core or by each user in its private installation. Each of these command classes implements a help function that exposes all expected or accepted arguments.

From the local host this component can accept any number of connections expecting a very simple text-based protocol. The actual front-end can then be anything from a plain terminal (like telnet) to a custom-made shell or (on a slightly different implementation) ROOT with its TAliEnGrid [9] wrapper. After passing the shared secret, further lines are interpreted as full commands to be executed and parsed as described above. Commands are processed sequentially on the same connection, but concurrent connections are not blocking each other so different application or shell instances can operate independently.

Included in the package, a custom shell component called Jsh provides the expected command line editing, history and tab-completion of commands. Jsh verifies upon startup if a JBox service is available and otherwise triggers the according service to start in background, while interacting with the user to supply the certificate's password in order to allow the initialization of the upstream connection within the service.

3. Standalone Grid middleware for certified Grid jobs
Beyond the integration of jAliEn in AliEn as a client component for Grid users and jobs, jAliEn was envisioned as a complete future Grid middleware, based on the AliEn schema of databases and directory services. In [10][11], these capabilities of jAliEn providing certified Grid jobs and files based on a new pilot job mechanism using cascaded digital signatures were presented.

4. Performance
As a reference point for the mechanism's scalability, the presented communication framework was benchmarked based on LFN queries to the file catalogue, equivalent to the information necessary to request for a simple directory listing. The testing involved the full stack of mutually authenticated and encrypted communication and object serialization.

The performed test cases were established using three test machines: one machine (Intel Core 2 Duo CPU 2.80GHz, 8GB RAM, solid-state drive) providing all central Grid service instances, including databases and a directory service, establishing a fully independent lab test environment. And two machines as Grid clients '*clientA*' (Intel i5 CPU 2.50GHz, 8GB RAM, solid-state drive) and '*clientB*' (Intel Core 2 Duo CPU 1.86GHz, 4GB RAM, solid-state drive). In three test cases, both machines executed 20 concurrent threads, each querying for all LFNs within a certain directory in a loop, each thread simply issued the same queries one after another, for duration of exactly 10 minutes. The queried directory contained 20 file entries (LFNs), therefore a client received all information existent within the catalogue concerning a certain LFN entry (e.g. parent and child node IDs, owner and permissions, file checksum, date of creation, size, physical replicas etc) as a whole. For each directory request the wall time was taken, while ignoring any processing or displaying of the respective information on client side and without any caching mechanisms in place since the purpose was to load the central services as much as possible.

Figure 2 below shows the results of the first test case with 20 concurrent threads per machine, based on 2 operating system processes with 10 internal threads each, all of which are requesting the respective LFNs in loop. In Figures 3 and 4 the same logic is used to stress the central services from two machines, on a larger scale. Figure 3 shows the response time with 300 concurrent threads per machine respectively 500 concurrent threads in Figure 4. In both cases there were 50 threads per client process instance, so 6 and respective 10 processes as client instances. In all test cases, the connection to the central service is established once per process, accordingly all internal threads of a process share one pre-established connection. This scenario mimics the JSite component, since the component has a single uplink and aggregates requests from locally executed Grid user interfaces or jobs.

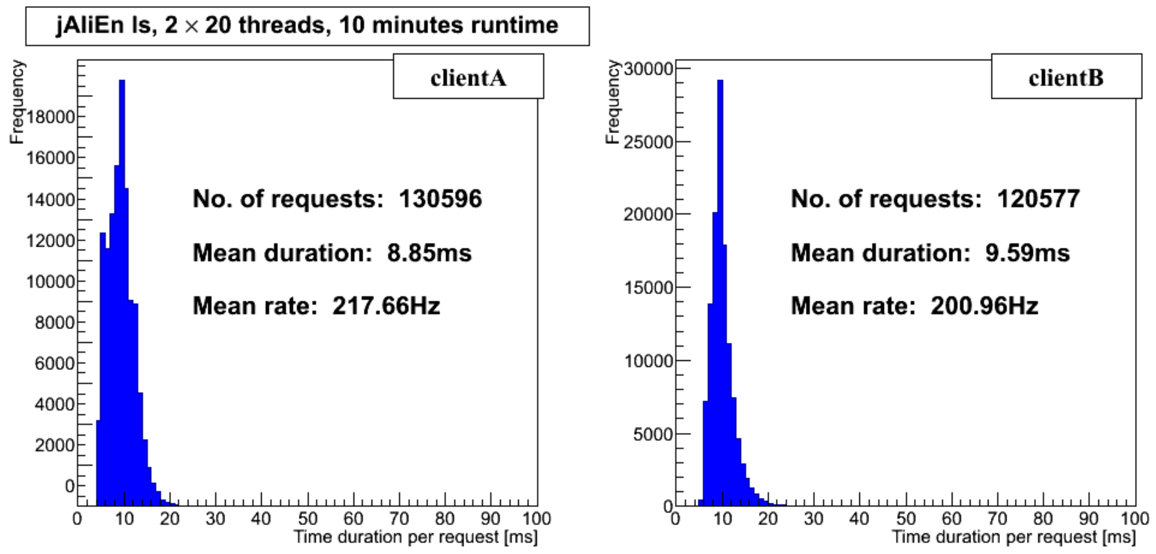

Figure 2. Two machines with each 20 concurrent requests for directory entries in loop.

Concerning the first case with 20 concurrent request loops per machine, the two clients were each able to receive more than 120k request responses successfully with a mean time of less than 10 milliseconds.

In the second test case with 300 requesting threads, the mean time for a request was approx. 27 milliseconds and the bulk of requests stayed below 90 milliseconds. In the third test case with 500 threads, the average request time was approx. 44 milliseconds and the bulk of requests were executed in less than 150 milliseconds.

In all three cases the jAliEn central service instance was able to serve requests at a rate of more than 400Hz (418.62Hz, 436.59Hz and 431.91Hz, which are the combined frequencies from the two clients that were run simultaneously in each test). As a reference point for comparison: the busiest AliEn central service API instance serves on average requests at approx. 250Hz and up to 400Hz during a peak load period with more than 50k concurrently active Grid jobs.

Despite the test setup using standard non-high-performance hardware and the fact that all relevant central service instances in the test environment were running on one single system, it was still able to match the performance of the production servers (HP DL380 G8 servers with 24 cores each).

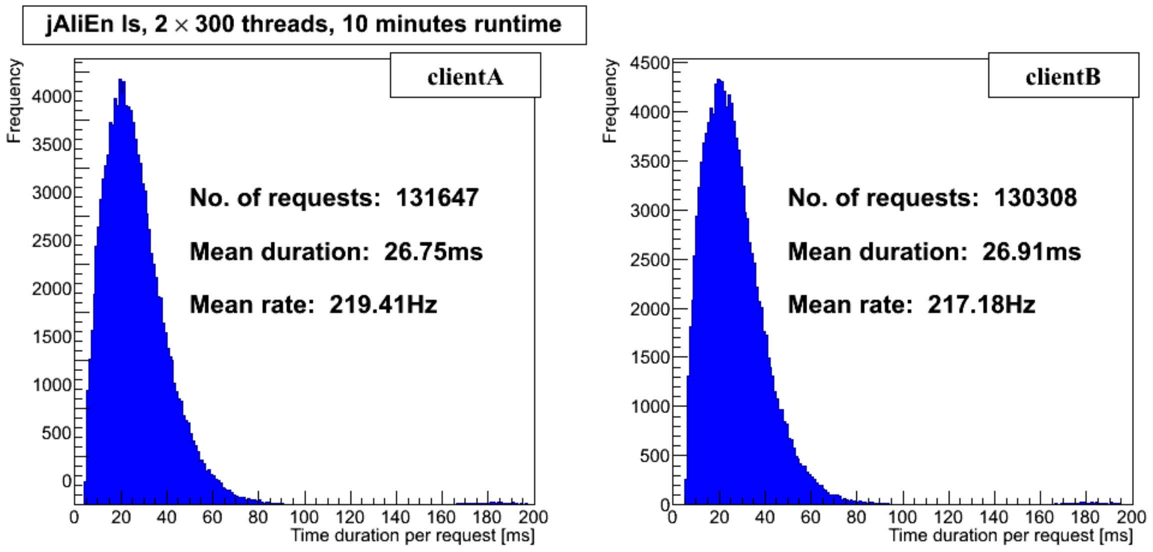

Figure 3. Two machines with each 300 concurrent requests for directory entries in loop.

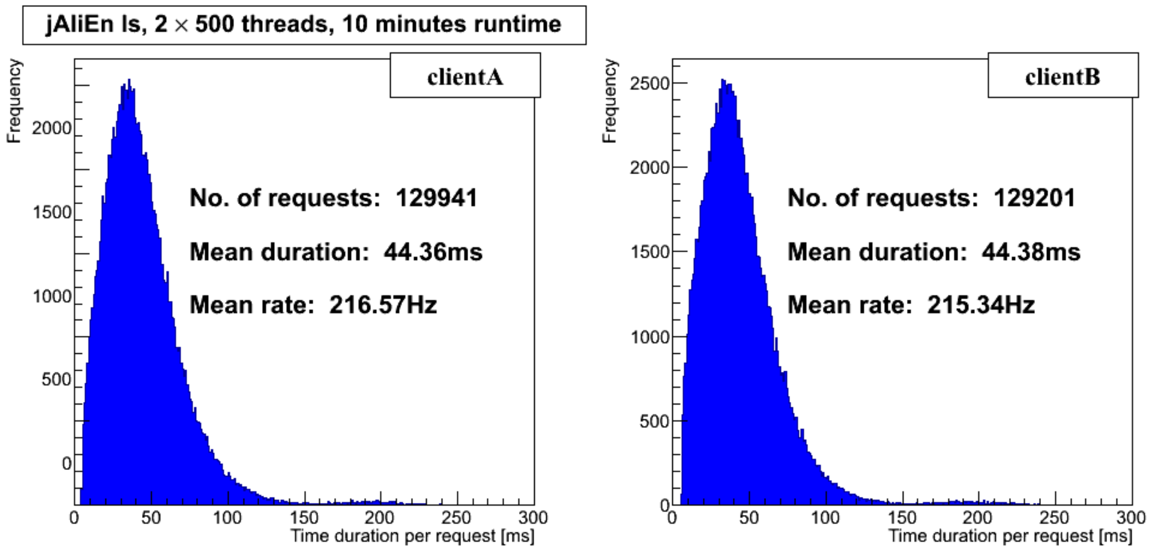

Figure 4. Two machines with each 500 concurrent requests for directory entries in loop.

5. Summary

In this paper we have presented the architecture, main features and performance tests of the new implementation of communication channels in jAliEn. The new middleware approach written in Java using serialized objects is highly efficient and will allow the system to run more jobs in the future.

Due to the utilization of only mutually authenticated and fully encrypted communication while providing single sign-on functionality based on only X.509 certificates (and thereby setting aside X.509 proxy certificates) significant security improvements concerning Grid client and service systems as well as Grid credentials can be achieved. Finally, jAliEn combines these security features with a high-level object-driven Grid communication protocol and a huge potential in terms of performance and scalability.

6. References

[1] ALICE - A Large Ion Collider Experiment. http://aliceinfo.cern.ch/ .

[2] AliEn. http://alien2.cern.ch/ .

[3] jAliEn. http://jalien.cern.ch/ .

[4] Oracle Corporation. Java Remote Method Invocation – Distributed Computing for Java http://www.oracle.com/technetwork/java/javase/tech/index-jsp-138781.html .

[5] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.

[6] The Legion of the Bouncy Castle. Bouncy Castle API. http://www.bouncycastle.org/ .

[7] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008.

[8] S. Tuecke, V. Welch, D. Engert, L. Pearlman, and M. Thompson. Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile. RFC 3820 (Proposed Standard), June 2004.

[9] Andreas Joachim Peters. AliEn Grid User Reference Guide V1.0. http://project-arda-dev.web.cern.ch/project-arda-dev/alice/apiservice/ .

[10] Steffen Schreiner, Costin Grigoras, Alina Grigoras, Latchezar Betev and Johannes Buchmann. A security architecture for the ALICE Grid Services. Proceedings of Science (PoS), The International Symposium on Grids and Clouds, ISGC2012:027, 2012.

[11] Steffen Schreiner, Costin Grigoras, Maarten Litmaath, Latchezar Betev and Johannes Buchmann. Certified Grid Job Submission in the ALICE Grid Services. Open Access Journal of Physics: Conference Series (JPCS), vol. 396, part 3, 032096, 2012.