CERN - ALICE Offline

Scalable Global Grid catalogue for Run3 and beyond

Project Overview

The AliEn file catalogue is a global unique namespace providing mapping between a UNIX-like logical name structure and the corresponding physical files distributed over 80 storage elements worldwide. Powerful search tools and hierarchical metadata information are integral parts of the system and are used by the Grid jobs as well as local users to store and access all files on the Grid storage elements.

In production since 2005, its current size is more than 2 billion logical file names. Factor of 20 increase of the namespace is anticipated with the ALICE upgrade in Run3, starting in 2020.. The larger namespace will be coupled with proportional increase of the access frequency by the tasks running on the Grid.

The current catalogue back-end is a traditional set of relational databases - MySQL RDBMS in master-slave configuration. This solution proved to be sufficient for smooth growth and fast access up to its present size. To scale the catalogue to the expected Run3 volume and access frequency, we are looking at new DB technologies for the back-end and in parallel at simplifying the catalogue schema while keeping the functionality intact.

This project paper describes the architectural evolution of the system, technology evaluation, benchmark results and conclusions.

Catalogue parameters

The current catalogue grew on a ratio of x1.62 between July 2015 and July 2016, and x2.65 in the last 3 years. The growth has shown to be smooth and constant, with some periods where there are peaks in the file registrations, specially during Pb-Pb data taking and processing. The current number of logical files surpasses 2 billion while the physical files reach almost 3 billion.

When it comes to performance, reads come as the most demanded operation, but the rest of operations are significant as well.

Command rates on aliendb06c.cern.ch

	Series	Last value	Min	Avg	Max	Total
1.	DELETE	162.8	0	314.2	6799	4922493099
2.	INSERT	121.2	0	162.2	4747	2541084131
3.	INSERT SELECT	66.16	0	122.8	1711	1923939923
4.	LOCK TABLES	0.082	0	0.076	2.508	1191399
5.	REPLACE	57.53	0	81.99	956.7	1284500619
6.	SELECT	4919	0	5042	23333	78990018178
7.	SET OPTION	65.64	0	664.8	21260	10414275605
8.	UPDATE	174.3	0	237	2339	3712309195
9.	Connections	15.9	0	11.39	137.5	178413557
	Total	**5582**		**6636**		**103968225710**

Figure 1. MySQL command rates between January and July 2016

These numbers represent the queries arriving to MySQL directly, but in fact we make use of a cache layer, that filters a fraction of the read-only queries, to reduce the load on the server while

giving faster responses. In particular, the cache deals with file information and locations (*whereis*), lookups for files in the hierarchy (*find*) and permission requests (*access*).

Some of these operations can be very heavy given the size of the catalogue and the iteration through many paths in it.

In addition, daily backups are kept using a slave server that stays in sync with the master.

Milestones

1. CHEP-2016

   Have a well-defined solution, architectural design and clarify features it will provide. Development of all these as far as it can go given the time period available.

2. Run 2 (after point 1.)

   Application workflow, possibly through UML or similar schemas. First implementation iterations, tests + initial benchmarking. Continue iterative development.

3. Run 3

   Full implementation ready for production, from DB backends to middleware frameworks.

Description

CVMFS pull-based solution through Apache Cassandra

The idea consists in having Cassandra as main DB backend handling the ALICE Catalogue information, which means re-implementing the hierarchy on the NoSQL data model paradigm. In order to achieve this, the application workflow (query analysis) will be combined with the conceptual data model into the logical data schema, and implemented in the physical data model. For end-users, a CVMFS interface will offer a browse-able catalogue with a familiar file system view of the contents. Any virtual organization could use such an interface as their own catalogue browser. On client demand, snapshots of the Cassandra database will be generated and distributed to these clients using the existing well-known and reliable CVMFS infrastructure, allowing for caching of the content along the way. Accessing the files requires interaction with the user credentials and the authentication/authorization services specific to each virtual organization, while keeping a common interface to access file metadata, which is one of the main goals of the project. In addition, we study the utilization of new fast memory technologies i.e. Intel Optane, providing yet another type of cache layer to store objects in its DRAM form, and/or persistent storage in its SSD form.

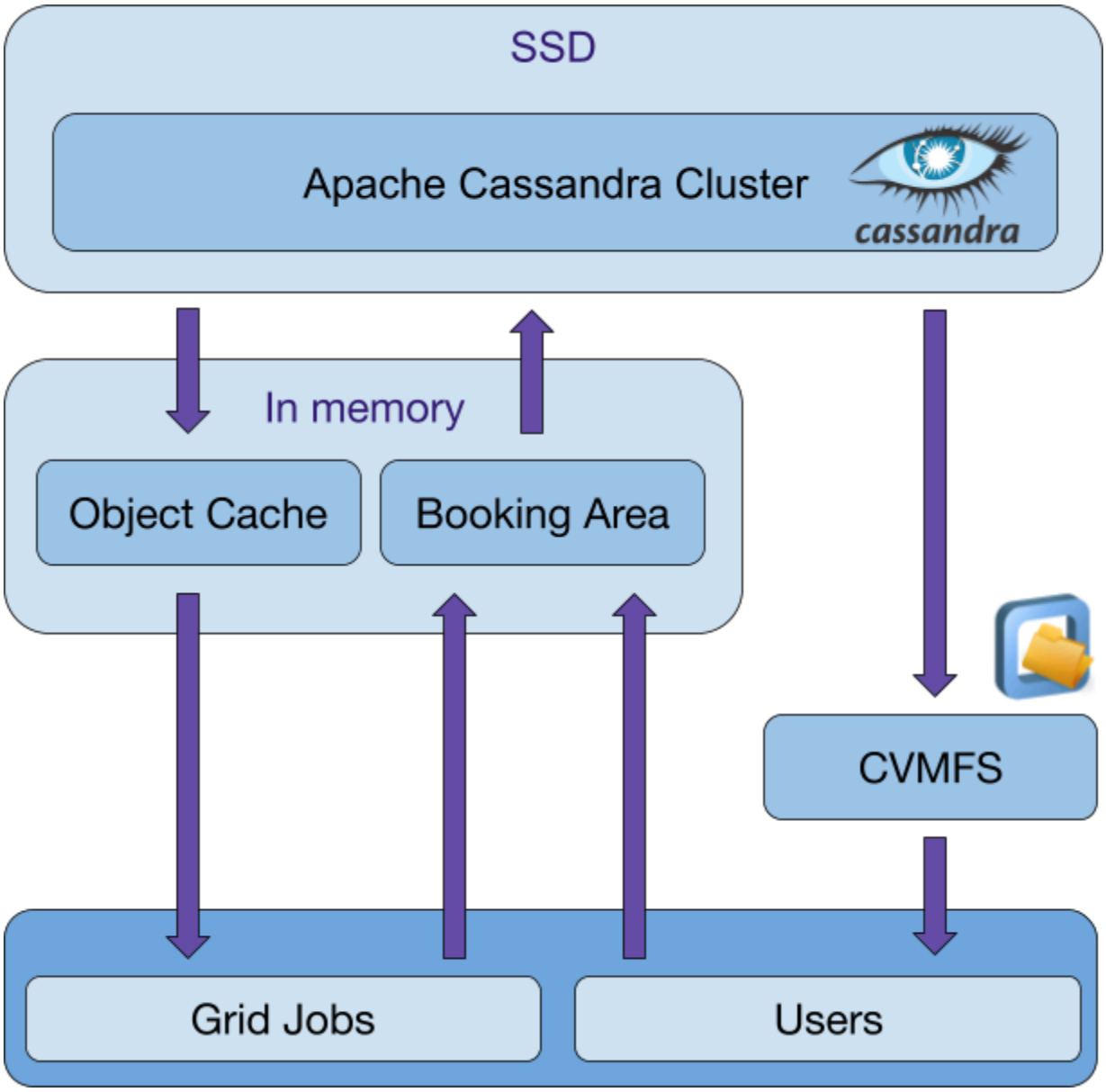


Figure 2. Project components diagram

Workflow explanation

In this section, we will explain in more detail how the components relate to each other.

On one end, we have the entities that make use of the services, thus needing database information, in this case, grid jobs and users. Grid jobs have a set of input files that normally consist of macros, OCDB snapshots, other code files or in general any file that the submitter wants to add to the job. In addition, jobs that do analysis will access experiment data from the analysis framework. In our project schema, the metadata information needed as well as the authorization envelopes (keys to be allowed to read/write from/to storage elements) will be prepared by the central services while processing the job insertions into the TaskQueue (grid jobs queue) and stored in a table, on a per waiting job basis, so that it can be queried quickly. It will be as well cacheable content, and will be sorted when the job starts, depending on the site it started on (for efficiency we usually run jobs where their data is). Of course, sometimes extra files will be used by jobs, such as OCDB entries, so the full mechanism that we currently support will still work as well, meaning that any job can ask for any file and be returned the list of locations optimally sorted related to its own location, it does not matter where the file is in the file hierarchy. Standard users will fit in the use-case just mentioned. However, they will use CVMFS to have a UNIX-like view of the ALICE Catalogue. Snapshots of the hierarchies will come from upstream on demand, querying the Cassandra database backend. We generate sqlite content that will be sent to the squid server next to the CVMFS client, and this client will retrieve the data from it, finally returning to the user the desired information. The process will appear transparent to the end user. CVMFS communication to squid and upstream is based on the HTTP protocol, which helps avoid conflicts with firewalls.

When new files are to be added to the catalogue, first a booking of their logical file names is made. We will use the Booking Area for this, which must be non-volatile, since there has to be

consistency with the database, in contrast to the Object Cache, that can go away without causing consistency issues. Once the initial booking is done, the client uploads the files to the corresponding storage element(s). To achieve this, the client needs to ask the central services for permission: check the user's file quota, find the best location based on storage element metrics such as space utilization, geographical position and network data, creation of envelopes with internal paths, MD5 checksums and more. When access is granted, the client uses the information provided by the central services, including the envelope, to trigger the remote copy and verifies that the file in the final destination is correct. An envelope is encrypted with the central services private key, while storage elements have the corresponding public key to decrypt the envelope and thus verify the legitimacy of the requested operation. Once the operation is finished, the final commit to the catalogue happens and the Booking Area reservation is released. In some cases, the final commit will depend on certain events to happen. For example, for jobs finishing in certain errors some files will be kept for a while, but by default not registered in the catalogue, unless the person in charge of the jobs needs them, normally for debugging. The Booking Area will be used for deletion of files from storage as well.

Lastly, at the other end, we have the central services themselves, in charge of providing all the information and operational support that the clients need. They use the database to perform multiple operations: catalogue maintenance, splitting jobs according to various strategies, calculate and refresh quotas, software packages management and more. The grid framework will query Cassandra directly, and in some cases interact with the cache. There are known use-cases where we know some values used by central processes will soon be used by jobs.

Use-case example: user reads a file vs job reads a file

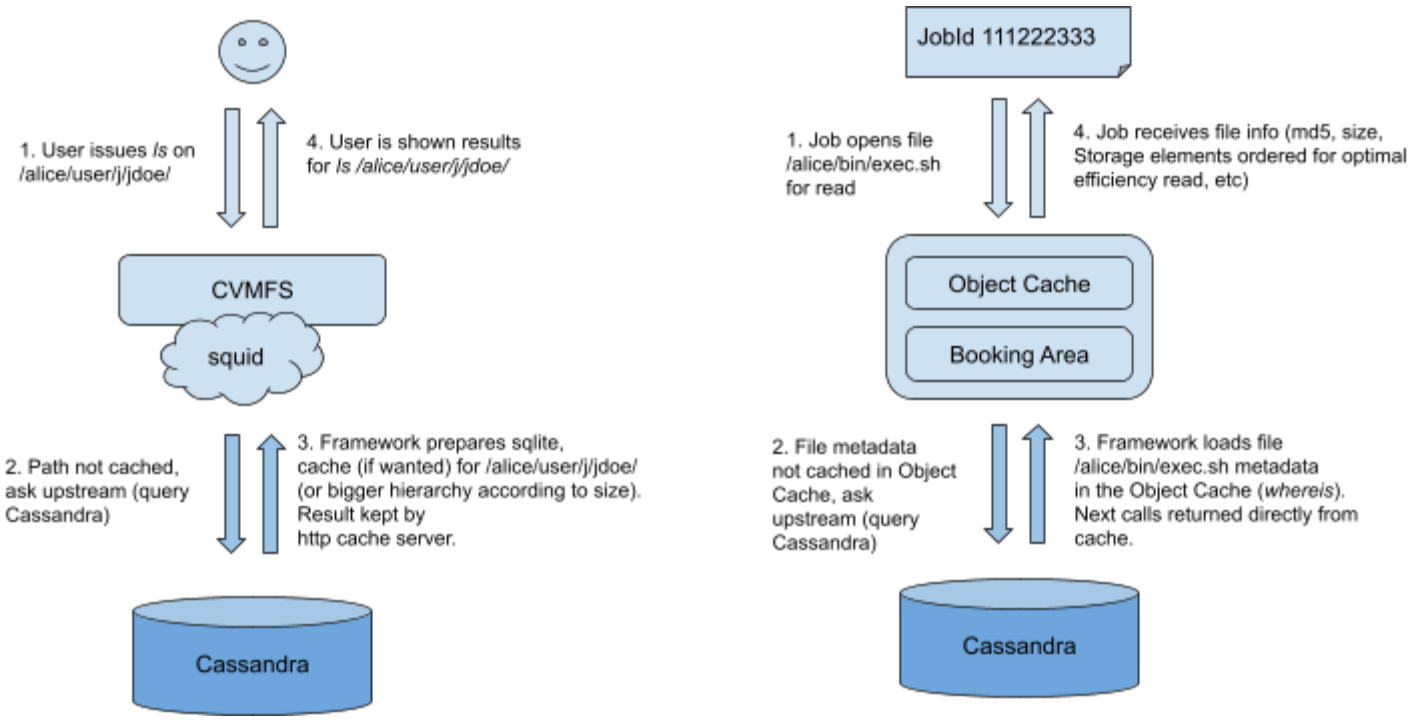
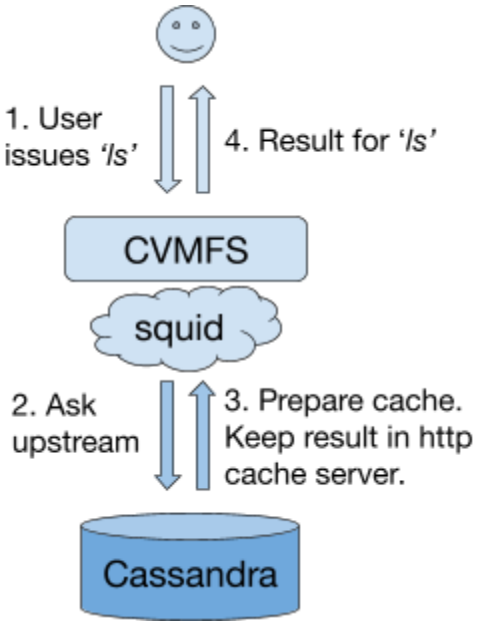
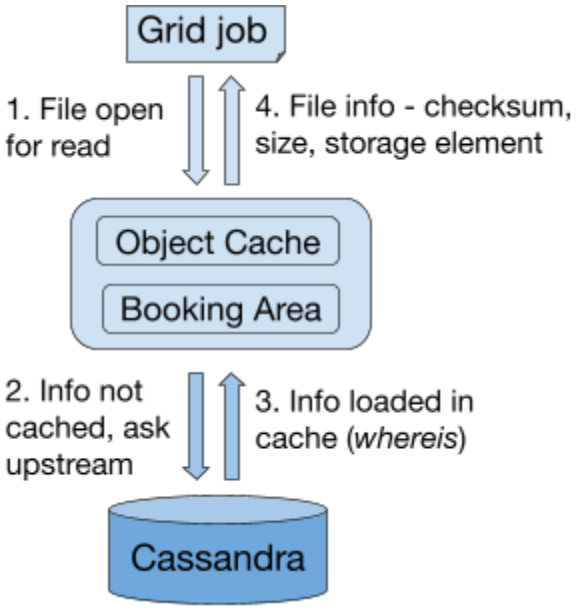


Figure 3. User and jobs read


1. User issues 'ls'
4. Result for 'ls'
CVMFS
squid
2. Ask upstream
3. Prepare cache. Keep result in http cache server.
Cassandra


Grid job
1. File open for read
4. File info - checksum, size, storage element
Object Cache
Booking Area
2. Info not cached, ask upstream
3. Info loaded in cache (whereis)
Cassandra

Specifications

1. Apache Cassandra

Cassandra is a free and open-source distributed database for managing large amounts of structured data across many servers. It was first developed by Facebook based on Google's BigTable and Amazon's Dynamo. It provides what other RDBMS or even NoSQL DBs can't: continuous availability, linear scale performance, operational simplicity and easy data distribution across multiple data centers or clouds, while being a highly available service and having no single point of failure.

Besides, its architecture is designed as a ring or circle shape, and replicates data across multiple nodes, following a masterless paradigm, and allowing failed nodes to be replaced with no downtime. This moves in a different direction than the traditional master-slave configuration or a manual and difficult-to-maintain sharded architecture.

From the technical point of view, the main challenge will be to migrate from a RDBMS schema to the new Cassandra model, based on the combination of application workflow and conceptual data model. In this type of NoSQL systems, the final implementation relies on the queries you need in your application. AliEn has a more or less constrained and well-known set of critical queries and the focus will be to optimize the DB model to have excellent performance in such queries, while keeping all the functionalities currently in the system.

In addition to the performance, we have to keep it mind that at the same time, we need to have consistent information provided to the clients. Cassandra offers different tunable levels of

consistency, that we will need to adjust to keep the information that needs to be read just after being written.

As a final note, we will study the use of the alternative to Cassandra named ScyllaDB. It is a fully compatible and drop-in replacement solution, so it should be easy to perform some benchmarking and comparisons with it. The public benchmarking and documentation assure very significant performance improvements compared to Cassandra, many orders of magnitude, and reaching 1M TPS per node. The technological difference resides mainly in the way to handle communications with the kernel, memory and core utilization. Scylla runs multiple engines, one per core, each with its own memory, CPU and multi-queue NIC.

2. Intel Optane

Intel Optane is the main breakthrough in memory design in more than 25 years. Its technology combines the revolutionary 3D XPoint non-volatile memory media with an advanced system memory controller. Intel's first benchmarks show a x8 data rate performance boost compared to their best NAND SSD on the market, while also having a much lower and stable latency (9 µs in their first tests) and x10 data density.

Intel announced they will provide both SSDs based on this technology, as well as DRAM DIMMs, both persistent. We consider the utilization of this new memory in its SSD form for persistent information storage (DB backend, Cassandra), since the total volume expected is big. For the cache and booking area space we consider the DRAM version, more performant but costly, where we do not need to support such big data spaces. The cache will contain objects, possibly in a key-value fashion, loaded on demand, to filter many queries away from the database. The content of such a cache can be lost without causing problems to the workflows. The booking area will keep data that is temporarily needed by the Grid framework while the final commit has

to wait for certain operations to happen, a kind of exchange area. It is relevant that in this case the memory is persistent, since it would allow the system to keep track of the files uploaded to Storage Elements at all times. Still, it would not be a major problem if sometimes the content were lost: affected jobs can just be rerun, while periodical consistency checks between the catalogue and the content of the Storage Elements will reveal mismatches to be rectified. Since the booking area is highly used by many concurrent tasks, having such a fast and common booking area will avoid synchronisation problems within Cassandra and speed up the whole file registration process.

3. CVMFS

The CernVM-File System provides a scalable, reliable and low-maintenance software distribution service. It is a well-known and spread tool in the WLCG and users are familiar with it. It is implemented as a POSIX read-only filesystem in user space (a FUSE module). Files and directories are hosted on standard web servers (backed by squid servers) and CernVM-FS uses outgoing HTTP connections only, thereby avoiding most of the firewall issues of other network file systems.

The aim of this project is to use the established CVMFS infrastructure to present users with a browse-able AliEn File Catalogue with the familiar view of a standard filesystem. In order to achieve this, there are currently some developments to be made: authentication and authorization within the CVMFS client and/or by catching system calls and overwriting client commands; a DB to sqlite snapshot creation plugin to load Cassandra content into CVMFS; and a more fine-grained way to manipulate CVMFS cached data.

Figure 4. CVMFS-DB communication

As shown in Figure 2, in principle we exclude grid jobs from using CVMFS for Catalogue exploration. The reason is that the grid framework will be modified in such a way that jobs will already have available all the metadata and authorization information for the files that they will use, on a per job basis, since we know beforehand the input files and analysis data they need. For particular usage of the hierarchy, such as *find* commands, the object cache or ultimately the DB will be consulted. Nonetheless, integrating grid jobs with the CVMFS catalogue a posteriori would be trivial since it would work in the same way as for users. Nonetheless, there is a potential cache turnover issue when having CVMFS used by jobs. It would be less or more significant depending on which data the jobs will need to query, and its volatility, but we have to assume the jobs could access any part of the catalogue. For example, analysis data will stay unchanged but macros or user files will change quickly over time.

As a last addition, we can consider the utilization of an sqlite hierarchy cache to deal with CVMFS calls, in case it appears useful, though in principle the impact of user operations on the overall system is small.

4. Other technicalities

Despite the fact that we want to explore fast memories for caching and better performance in general, we aim as much as possible to have an infrastructure that does not rely on the nature of the machines, being able to scale, as for example stated in Cassandra design, by adding more servers that in principle could be commodity ones, while having the possibility to easily migrate the whole system between machines of different nature.

In addition, the APIs developed at the different levels of Figure 2 should be flexible enough so that we do not have a hard constraint on the DB backend, since technologies including databases will keep evolving and we want to be able to adapt and improve accordingly. The utilization of CVMFS allows the project to provide a common data browsing interface that may also be used by other communities whose catalogues need to resemble a standard filesystem. Of course, particularities of their respective frameworks would have to be taken into account, to optimize the usage patterns and performance per case.