EPJ Web of Conferences **214**, 03037 (2019) https://doi.org/10.1051/epjconf/201921403037
CHEP 2018

JAliEn: the new ALICE high-performance and high-scalability Grid framework

M. Martinez Pedreira[1,*], *C. Grigoras*[1], and *V. Yurchenko*[1]

[1]CERN, CH-1211, Geneva 23, Switzerland

Abstract. The ALICE experiment will undergo extensive hardware and software upgrades for the LHC Run3. This translates in significant increase of the CPU and storage resources required for data processing, and at the same time the data access rates will grow linearly with the amount of resources. JAliEn (Java ALICE Environment) is the new Grid middleware designed to scale-out horizontally to fulfil the computing needs of the upgrade, and at the same time to modernize all parts of the distributed system software. This paper will present the architecture of the JAliEn framework, the technologies used and performance measurements. This work will also describe the next generation solution that will replace our main database backend, the AliEn File Catalogue. The catalogue is an integral part of the system, containing the metadata of all files written to the distributed Grid storage and also provides powerful search and data manipulation tools. As for JAliEn, the focus has been put onto horizontal scalability, with the aim to handle near exascale data volumes and order of magnitude more workload than the currently used Grid middleware. Lastly, this contribution will present how JAliEn manages the increased complexity of the tasks associated with the new ALICE data processing and analysis framework (ALFA) and multi-core environments.

1 Introduction

The ALICE [1] experiment at CERN is one of the four big particle detectors at the Large Hadron Collider (LHC). Its main goal is to study heavy-ion (Pb-Pb) collisions at high center of mass energies.

During a normal data taking year, ALICE creates and collects up to tens of petabytes of data. To store, process and analyse it, ALICE uses the Worldwide LHC Computing Grid (WLCG) [2], which unites 170 computing centres in 42 countries.

Java ALICE Environment (JAliEn) [3] is the new middleware framework used on top of WLCG within ALICE. It coordinates the computing operations and data management of the collaboration on the Grid, and is the evolution of the legacy framework called AliEn. Different architectural and technological changes have been introduced to JAliEn to make it capable of fulfilling the ALICE requirements in Run3 and beyond, and some will be described by this work.

[*] Corresponding author: miguel.martinez.pedreira@cern.ch

The JAliEn File Catalogue takes over from the previous MySQL-based file catalogue, used in AliEn. The catalogue annotates every file stored on the ALICE Grid distributed storage. As of 2018, it contains more than 5 billion logical entries and 12 billion entries overall, including the pointers to file physical locations. The catalogue growth over time is constant and smooth, however the number of files will increase significantly in Run3. The JAliEn catalogue has been developed and benchmarked extensively to assure it will meet the future needs. It makes use of new technologies, namely the Cassandra [4] NoSQL model. We present the details and performance tests in the next sections.

1.1 Evolution of computing resources in the past, present and future

From the initial Grid capacity to today, the deployed computing resources have grown continuously and in the LHC data taking years (2010 to 2018) the CPU and storage capacity have increased by a factor 10. It is expected to have such continuous growth in the future as well. The resource growth is illustrated in Figure 1 for storage, and it is similar for CPU. In addition, the ALICE Run 3 upgrade will translate yet into more resource increases associated to the O^2 facility, which is a purpose-built large computational cluster to be used for synchronous and asynchronous data processing. It will comprise of 60 petabytes of disk and around 100000 CPU cores.

For improved scalability and maintainability, the ALICE high-level Grid services and clients are being reimplemented as the new JAliEn framework.

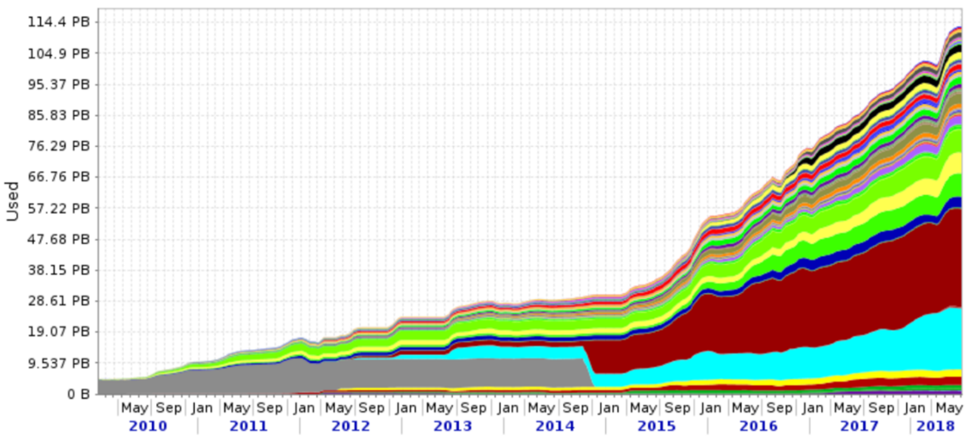

Figure 1. Data accumulation in ALICE from 2011 to 2018, each colour corresponds to a storage element

2 Java ALICE Environment framework

JAliEn consists of a set of services with different functions. Firstly, we have the central services, called jCentral. Their authentication and authorization mechanisms are discussed further below.

jCentral takes care of the job queue, the TaskQueue. The functions related to the TaskQueue are serving job submissions, job splitting into smaller sub-tasks, calculation and control of job quotas per user, handling the information and parameters of the different computing centres (CPU resources), registering job outputs, job status transitions, job matching and job tracing.

jCentral is also responsible for the data management. It takes decisions on file placement and file source for all Grid jobs, depending on several factors such as client location (being a client any entity connecting to the service, such as a job or a user) and the status of storage and network topology between the client and the storage servers. A filesystem-like hierarchy is kept within the File Catalogue and offered to the users, so they can run shell commands and use powerful lookup tools. Moreover, data transfer services run centrally and are used for balancing storages, data replication and data recovery.

The site services have the function of interfacing jCentral and the computing resources available on every site. A frontend machine called VoBox runs a service that deals with the local resource management system of each site on behalf of jCentral. It controls the submission of new jobs and provides an aggregation point for the services and monitoring messages from the jobs running on the site and from other services running at the site. The software package distribution on the worker nodes is done via CernVM-FS [5]. It is a reliable, low-maintenance distribution service implemented as a POSIX read-only file system in user space.

The users connect to JAliEn using their standard Grid certificates and get a shell prompt like on a Linux console. From the shell they can run a variety of commands to add, move or delete files, submit jobs, or run find and list commands, among many others. Clients can connect from their own scripts or programs using the WebSocket [6] endpoint. Support for that protocol is standard in the relevant programming languages.

2.2 Authentication and authorization model

X509 [7] is a Public Key Infrastructure standard that specifies, among other things, formats for public key certificates and a validation algorithm for the certification chain. The X509 model assumes a strict hierarchical system of Certification Authorities (CAs). JAliEn has its own CA and it is extensively used to create Token Certificates that will be described in detail in the next subsection.

2.2.1 Client-server

We extend the standard X509 model by introducing the 'Token Certificate'. It includes more fine-grained functions, like distinction between a JobAgent (pilot) and user payload - each token can only be used to execute a fixed set of operations. The authorization level is controlled by jCentral and checked on every request.

The additional functions are encoded in the DN and extensions of the X509 certificate. This makes the software compatible with the current X509 security libraries while adding the flexibility of mapping roles and capabilities to the tokens.

A utility has been developed as well to renew tokens automatically. This is important for long-lived services on the site frontend machines (VoBoxes).

An example of a Token Certificate for a payload is given below:

Subject: OU="queueid=1038905674/resubmission=3/user=mmmartin", OU=jobagent, CN=jobagent, CN=Jobs, O=AliEn, C=ch
Issuer: JAliEn-CS

With this token, the JobAgent can operate on job 1038905674 as user 'mmmartin' with the full rights of this user. The resubmission field shows how many times the job was executed, and that previous tokens (for resubmissions 0-2) are invalidated by jCentral.

2.2.2 Data access

JAliEn implements the data access model used in the ALICE Grid for the last decade. In this access workflow, every time a client interacts with a storage element (for read/write/delete operations), it requests permission from jCentral. The latter creates an "envelope" (a token), that contains metadata of the file in question, plus a signature generated with the central services private key. The envelope is encrypted using the public key of the destination storage element (each storage has its own keyset). This way, only the destination storage will be able to decrypt the envelope (using its private key) and will verify that the content is correct. The keys mentioned in this workflow are standard X509 key-pairs. Each envelope is unique for an operation, a file and a user, and has an expiration time.

2.3 Job pilot

The job pilot (called JobAgent) executes user payloads assigned by jCentral. In JAliEn, the implementation of the JobAgent is split in two, following its two main logical functions.

The first component is the base JobAgent equipped with a Pilot Token Certificate. Such a credential allows the JobAgent to do job matching, which consists of reporting the worker node conditions and asking jCentral for a matching user payload. The reply is based on the JobAgent conditions report and general information about the site already available to jCentral. It contains the job id, the job description and a Job Token Certificate, which works as explained at the end of section 2.2.1. The JobAgent also controls and monitors the payload: memory, disk and CPU usage. The metrics are sent back to jCentral to keep the job database up to date and mark payload execution status transitions.

High Performance Computing environments are becoming a useful opportunistic resource and typically offer full computing nodes instead of single CPU cores for payload execution. In these cases, the JobAgent instantiates and controls several user payloads (see next paragraph) and does a more refined job matching.

The second component is the JobWrapper. It handles several aspects based on the payload requirements: download input files, create the job sandbox, set the environment of the task, launch its execution, and at the end of execution, upload the output files (including archives) and manage their registration in the JAliEn catalogue. The JobWrapper is instantiated from the JobAgent. The JobAgent thus has an exclusive communication channel with the JobWrapper through Java stdin/stdout pipes, used e.g. to transfer the credentials of the task to the JobWrapper.

2.3.1 Isolation of the payload using containers

Critical security aspects of the execution of user payloads and control of the execution environment have been addressed with the use of containers. The JobWrapper adopts containers in a solution-agnostic fashion. Non-containerized tasks could potentially try to steal others tasks' credentials or the ones from the pilots. Even if JAliEn minimizes the risks with the introduction of the Token Certificates and the sharing of credentials through the handles in Java, containers further enforce isolation. Strict sandboxing is also necessary to make sure a task hasn't left any malign code to the next one and to avoid resource misuse. The JobWrapper leverages the control groups (cgroups) capabilities of the container engines to limit the CPU, memory and disk available to the payload, and the namespace capabilities to limit the filesystem operations to the local scratch space.

3 JAliEn File Catalogue

The current AliEn File Catalogue is based on MySQL with master-multislave configuration. The slaves are in sync with the master and are used for backups and hot standby. The master is the only entry point for queries modifying the catalogue content and for read operation, since fully guaranteed consistency is a requirement. This architecture has several critical limitations: single point of failure (unique master), low-availability design (partially solved by the slaves), master relying on powerful hardware and manual sharding.

The Cassandra model has a completely different architecture (that resembles a ring) and solves the aforementioned limitations of MySQL: high-availability and no single point of failure are guaranteed by having many master machines with replicas; horizontal scalability can be achieved by adding more machines to the cluster; consistency can be tuned on different levels.

3.1 Performance and size goals

The current query rates and data volume handled by the MySQL catalogue implementation are taken into account to estimate future needs: 15000 queries per second and 110 petabytes hosted storage respectively. Additional 60 petabytes and 100000 cores will be available in the O^2 facility and about 25% yearly increase of resources must be handled as well. Given this scenario, the minimum performance goal has been set to sustain 100000 operations per second on a 50 billion entries namespace. However, Cassandra allows the system to adapt dynamically and scale up or down when the needs change.

3.2 Data model

The performance and adaptability of the database rely critically on the data model. Table 1 shows the various data model elements foreseen for the JAliEn catalogue.

Table 1. Column families of the model, primary keys in italic

index	*path_id*	*path*	*ctime*	child_id	flag	
metadata	*path_id*	*id*	ctime	owner	gowner	size
	type	perm	jobid	checksum	metadata	pfns
ids	*child_id*	path_id	path	ctime	flag	
se_lookup	*senumber*	*modulo*	*id*	owner	size	

The **index** column family allows exploring the hierarchical structure and getting the id of a logical filename or all logical filenames in a folder. It will receive many queries but the content is highly cacheable. The **metadata** table will be queried using the data retrieved from **index**. It contains file system-like metadata. The **ids** table provides the inverse functionality of the **index**: allows going from an id to the logical filename. This is especially useful to know which files are in a given storage element, where the physical names are id based. The **se_lookup** column family is used to account the content of the storage elements and calculate the quotas for the users (total space and number of files).

SizeTieredCompaction, compression and replication factor 3 are set for all column families.

3.3 Apache Cassandra and ScyllaDB internals and tuning

These two technologies provide mostly the same base features and currently are compatible at data block level, thus the application layer is compatible with both without changes, making a migration from one to another a straightforward process.

Apache Cassandra runs as any Java application. However, there are some tricks in the implementation that allow for complex memory manipulation by using the Java Native Interface framework. Extensive tuning of kernel, disk, network, CPU and memory parameters is needed to allow Cassandra to run to the full extent of its capabilities. The disk tuning makes the OS aware of the solid-state disks, and adapt to the nature of the database operations, for example read-ahead and avoiding reading extra bytes when possible, deadline scheduler, and number of requests. For the CPU, scaling is disabled, so it always runs on the highest frequency without transitions. The modified network parameters allow the service to profit from increased buffer sizes, faster time-wait recycle, window scaling and an increase of both the number and backlog of incoming connections. Lastly, swap is disabled.

Cassandra's startup script optimizes a set of JVM parameters. The G1GC garbage collector is used for the tests, with increased memory (16GB). Some extra GC parameters establish the triggers and thresholds for the garbage collection operation, to avoid stop-the-world pauses.

Finally, the Cassandra configuration is modified to enhance the concurrency of reads and writes with much higher values than those shared in the community. The number of memory table writers and the size of the key cache are increased. The modifications to the Cassandra base configuration are the ones that made the most impact on the overall performance of the database. The total gain factor after all the tuning efforts is 5, which is quite significant as it reduces the number of nodes required by the same factor. Even fully optimized, Cassandra didn't manage to saturate the system resources in any benchmark and after some investigation, the conclusion was that there are locks on the application level that cannot be made faster or be avoided.

The lack of resource saturation led us to run tests also on ScyllaDB [8]. It has a completely different internal implementation, is written in C++ and splits into one database engine per core, instead of using threads. The memory of the machine is distributed among those database engines, called shards. ScyllaDB discovers the OS and hardware of the host and automatically does the tuning of the kernel and file systems. It also runs fully asynchronous operations and bypasses most of the system calls.

3.4 Benchmark results

The test cluster is a 6-server ring, with 2 Xeon E5-2660 (56 hyperthreaded cores total), 256GB of RAM and a Samsung 850 Pro SSD SATA each. The links between server machines and client machines are routed through a dedicated 40G switch, with 10G for each individual machine.

Two benchmarks have been used. Firstly, the "cassandra-stress" tool, which is included in all Cassandra and ScyllaDB distributions and provides options to run different workloads and distributions: population, read-only, mixed read-write, using Gaussian, exponential or other distributions. One can also tune the requests per second per client and other options. A positive aspect of this tool is that the results can be used to compare different setups against community-supplied results. The second benchmark is the "JAliEn benchmark", which implements all the file catalogue functionalities and provides the same functions as in the cassandra-stress suite. The results of the JAliEn benchmark are more relevant since they represent more realistically the performance for the ALICE use-case. Figure 2 shows the results of the various tests done with "cassandra-stress" on our test cluster.

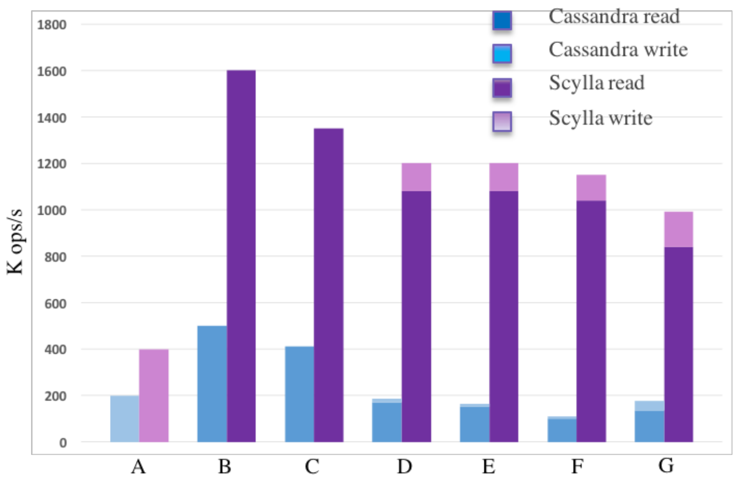

Figure 2. Cassandra-stress results for ScyllaDB and Apache Cassandra. The synopsis of each test is given in the text.

The first workload (A) populates 5 billion entries. The rest use a Gaussian distribution with the whole namespace, mean on the centre and different standard deviations. B and C do read-only with 10K and 1M standard deviation. D, E and F run a mixed (10 times read for 1 write) workload with standard deviations of 100K, 1M and 10M respectively. G runs a similar workload as D with the difference that instead of controlling the mixed operations from the application, it is done with the number of threads instantiated for each type of operation. There was no resource saturation on Cassandra nodes in any test. ScyllaDB had 100% I/O during population and 100% real CPU utilization (excluding polling) in the rest of the tests, which shows the more efficient hardware use of this backend.

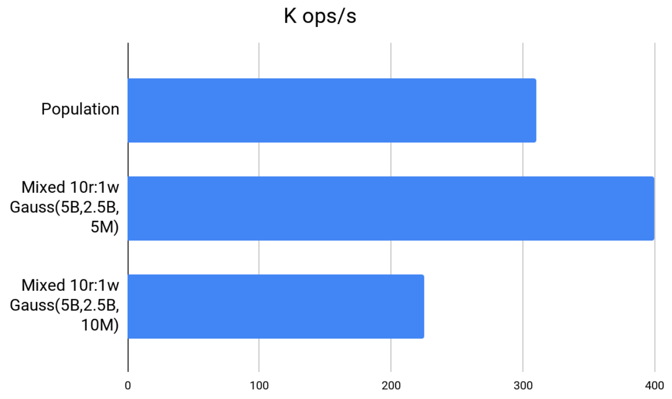

Figure 3. JAliEn benchmark results for ScyllaDB

As shown by Figure 3, the results obtained by the new schema fulfil the initial requirements for the File Catalogue in Run3. The bottom benchmark result, that has a standard deviation of 10M, is the closest model to our current production workload. To reach this conclusion, different metrics and query information were gathered from the File Catalogue in different periods. Disk utilization was 100% in all three tests.

3.5 Lessons learnt

There are some database maintenance operations that were not mentioned so far but need to be run periodically. They compact database entries and synchronize the content of the different instances, resulting in significant I/O and thus affecting performance.

Running several Cassandra instances per node is possible, in order to maximize resource utilization. While the cassandra-stress tests run with dual instances revealed a 30-40% gain, setup complexity increases as well and affects different Cassandra tools and commands.

Monitoring is of key importance not only to understand the behaviour of the database and workloads, but also to detect bottlenecks, for example in hardware. In the JAliEn benchmark (Figure 3), the disk is saturated. An easy improvement with large performance impact is to switch from SATA SSDs to NVMe disks. The PCI-express communication provides much higher bandwidths and IOPS. At the same time, monitoring will be helpful to find out if there is a good balance between disk, memory and CPU depending on the workload.

4 Conclusions

JAliEn and its new File Catalogue have been developed with focus on the scalability and performance to fulfil the Run3 requirements for ALICE. The new design includes modern and maintained technologies and concepts, and the development tracks the evolution and trends of the software and hardware. One example for the latter would be the Intel 3D X-Point memory, introduced to the market in 2017, for the Cassandra/ScyllaDB hosting servers. JAliEn development is consolidating and the framework is being put into production gradually. The File Catalogue migration will come after the deployment of JAliEn completes. Full database imports and stability of operations in production are still to be validated.

References

1. A. Abrahantes, K. Aamodt, R. Achenbach et al., *The ALICE experiment at the CERN LHC* (Journal of Instrumentation: 3(08): S08002, 2008)
2. I. Bird, P. Buncic et al., *Computing models of the WLCG* (CERN CDS: record 1695401, 2014)
3. A. G. Grigoras et al., *JAliEn: a new interface between AliEn jobs and the central services* (Journal of Physics: Conf. Ser. 523 012010, 2014)
4. A. Lakshman, P. Malik, *Cassandra: a decentralized structured storage system* (ACM SIGOPS ref. 1773922, 2010)
5. J. Blomer et al., *Distributing LHC application software and conditions databases using the CernVM file system* (Journal of Physics: Conf. Ser. 331 042003, 2011)
6. I. Fette, A. Melnikov, *The WebSocket protocol* (IETF RFC6455, 2011)
7. M. Cooper, P. Hesse et al., *X.509 Public Key Infrastructure* (IETF RFC4158, 2005)
8. A. Kivity, D. Laor et al., *Building the Real-Time Big Data Database (whitepaper)* (https://www.scylladb.com/resources/whitepapers, last read on 15 November, 2018)