

NoSQL technologies for the CMS Conditions Database

Roland Sipos for the CMS Collaboration

CERN, CH-1211 Geneve 23, Switzerland

E-mail: roland.sipos@cern.ch

Abstract. With the restart of the LHC in 2015, the growth of the CMS Conditions dataset will continue, therefore the need of consistent and highly available access to the Conditions makes a great cause to revisit different aspects of the current data storage solutions.

We present a study of alternative data storage backends for the Conditions Databases, by evaluating some of the most popular NoSQL databases to support a key-value representation of the CMS Conditions. The definition of the database infrastructure is based on the need of storing the conditions as BLOBs. Because of this, each condition can reach the size that may require special treatment (splitting) in these NoSQL databases. As big binary objects may be problematic in several database systems, and also to give an accurate baseline, a testing framework extension was implemented to measure the characteristics of the handling of arbitrary binary data in these databases. Based on the evaluation, prototypes of a document store, using a column-oriented and plain key-value store, are deployed. An adaption layer to access the backends in the CMS Offline software was developed to provide transparent support for these NoSQL databases in the CMS context. Additional data modelling approaches and considerations in the software layer, deployment and automatization of the databases are also covered in the research. In this paper we present the results of the evaluation as well as a performance comparison of the prototypes studied.

1. Introduction

The CMS Conditions Database[1] is responsible for storing alignment and calibration constants that record a given state of the CMS detector[2]. This data is essential for the analysis and reconstruction of the recorded data, and also a key element in the data processing stage. Therefore a proper synchronization is required in the data-flow. The status of the CMS detector is stored as a set of binary objects, serialized through the CMS software framework and stored as BLOBs[3] in an Oracle database provided by CERN IT department. This particular data type has the following characteristics:

- it grows with time and requires frequent lookups,
- the relational representation of the conditions does not involve table joins,
- it is free from table scans and the requested data is always accessed by a primary key,
- it lacks any kind of complex and nested query,
- transactions on the dataset may be unnecessary,
- conditions (BLOBs) are never updated, only added/inserted,
- the whole environment is generally read heavy,



- concurrent writes are absent.

Based on these observations, different non-relational data storing solutions may be applicable with greater convenience with regards to the current Oracle solution, and even with possible performance gains. Our main goal is to find and evaluate these alternative technologies for the CMS conditions for storing large binary objects, and its meta data with taking into account the considerations above.

2. Empirical evaluation

As a first step we excluded databases from the candidates that are inappropriate for our use-case. These characteristics mostly affects the underlying storage layer (like in-memory databases and only asynchronous writes for data persistence) or if the user interfaces and drivers doesn't have native support for C++ that is essential for CMSSW and for Java, which is required for our testing environment. After this pre-selection the next step includes the performance testing of the following three NoSQL databases, namely:

- *Cassandra*[4] *v2.0* - This distributed database aims for linear scalability and high availability on commodity hardware or cloud infrastructure. The data model is basically a hybrid between a key-value store and offers strong emphasis for denormalization and materialized views, and also supports built-in caching.
- *MongoDB*[5] *v2.6.9* - This document-oriented database has a support for horizontal scalability with its sharding solution, and also supports the GridFS infrastructure for storing large binary objects. With the combination of the two features, users have full control on the BLOBs data placement and their distributed access.
- *RIAK*[6] *v2.0* - This distributed database is designed for high-availability and operational simplicity, supporting a basic key-value data model in a log-structured hash table. Newer versions have *Apache Solr*[7] integration for advanced search capabilities.

2.1. Deployment

Several different setups were used as the base of the evaluation for the selected databases. For the deployment we used an automated Puppet environment to install, configure and launch the database clusters on virtual machines on the CERN OpenStack infrastructure. In the cooperation with CERN IT, we were able to use an OpenStack tenant exclusively, avoiding the overcommit of the resources and also preventing any user interference during the performance testing. The deployed setups are described on the following way:

- *Standalone* - This setup consists of a single database instance, deployed on *m1.large*¹ flavored VM with the image of an SLC6 Server (x86-64, built: 2014-08-05).
- *Large cluster* - This setup consists of five *m1.large* flavored instances with the same image as for the standalone setup.
- *Medium cluster* - This setup consists of four *m1.medium*² flavored instances with the same image as for the standalone setup.

The number of nodes of a given cluster setup is equally distributed on the underlying hypervisors, and during the testing phase only the participating nodes existed in the environment. The tenant's storage element had an optional SSD caching solution, so the testing were also done with that option in order to compare those results with spinning disk access.

¹ m1.large flavor - RAM: 8 GByte, VCPUs: 4, Ephemeral storage: 80 GByte

² m1.medium flavor - RAM: 4 GByte, VCPUs: 2, Ephemeral storage: 40 GByte

2.2. Testing framework

A unified extension (*CustomSamplers*) was implemented for the performance testing framework *JMeter*[8], that has a wide variety of support to write and read data in the databases. The extension has several utilities that handle the testing logic, the assignments of file requests to the worker processes and the result handling and propagation to the listeners. This package also includes a collection of tools for each database:

- *Deployers*: are responsible for building up the data model for the conditions,
- *QueryHandlers*: simulate the CMS workflow by accessing the binary objects,
- *ConfigElements*: configure and handle different persistency objects,
- *Samplers*: collect information about the queries, and report them to different JMeter listeners.

Test plans are XML configurations that set up the behavior of the testing engine by controlling thread numbers and their creation with the ramp up time. These also configure the connection layer and assign read requests to the created threads. With these XML files we aimed to maximize the throughput on the scale from 1 to 9 TPS (threads per second) where a thread's actual request time is the duration of two read operations to finish, defined by a query pair (see Table 1).

Table 1. The request time is the sum of two read operations.

Description	Request
Get Hash for (Tag, Since) pair	<pre>function GetHash(Tag, Since) Hash = Connection.IOVs.Get(Tag, Since) return Hash end function</pre>
Get Payload for Hash	<pre>function GetPayload(Hash) Payload = Connection.Payloads.Get(Hash) return Payload end function</pre>

A JMeter *Simple Data Writer* listener aggregates the results and writes a CSV output file from the samples. A given sample contains several different metrics of the corresponding measurement as follows:

- *Timestamp*: the creation time of the worker thread.
- *Thread name*: shows the ID of the thread.
- *Success*: either true for a successful request or false if there was an error.
- *Response code*: indicating if the request was successful.
- *Bytes*: size of the requested data (size of the payload).
- *Active threads*: the number of active threads by the time the sample was created.
- *Request time*: duration of the request pair to finish.

With this solution we can explore different limitations for saturating factors like network bandwidth, the access of persistency objects and the storage elements (ephemeral disk/SSD) for this particular BLOB access. These test plans can be categorized in the following way:

- *Standalone* - In this case the testing engine runs only on the master JMeter engine. With this test we aim to find the maximum threshold (threads per second) on the persistency objects, before facing networking limitations.
- *Remote* - During the remote tests 4 worker JMeter engines are participating in the testing, and there is a separate master engine that collects the reported samples from the worker nodes. With this test we can achieve the same throughput, without compromising the performance of the application layer.

The tester engines were not deployed in the same environment as the clusters' nodes, but rather on a separate tenant. This solution is feasible, as the JMeter processes do not cause interference with the databases during the testing period.

2.3. Test data

Based on several considerations we used a random generated binary test dataset, that consists of 2600 small files, with the sum of only ≈ 31 GByte with the distribution shown in Fig.1.

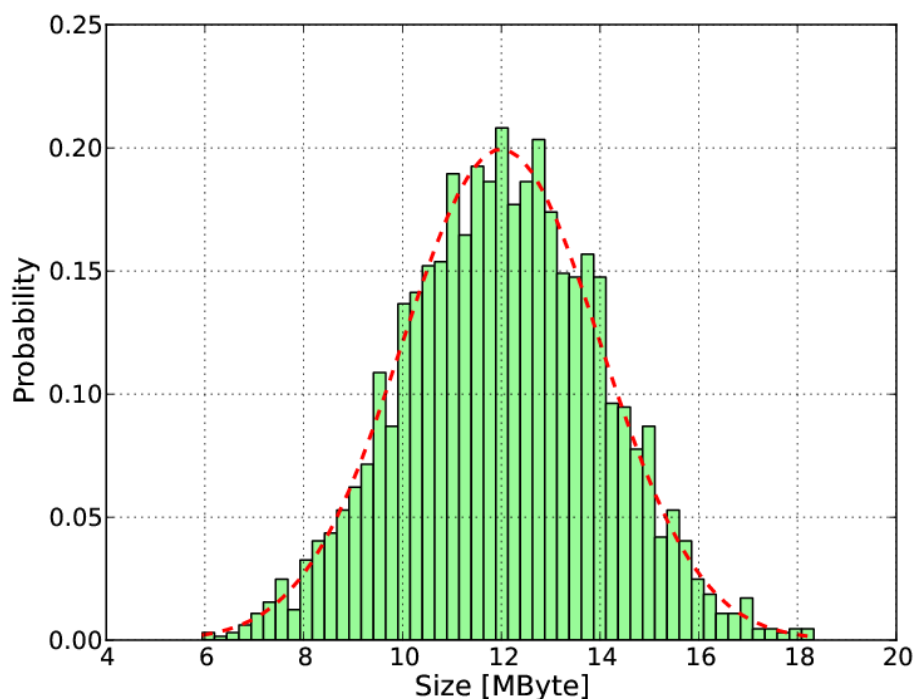


Figure 1. File-size distribution for the test dataset ($\mu = 12$, $\sigma = 2$).

Test data are written once and only in standalone mode with low throughput (0.5 TPS), followed by the execution of a burn-in test to identify any potential problems before the read heavy scenario. Automated monitoring of the clusters' nodes is integrated in the test plans, that is started at the beginning of every test, and finished with the collection of the monitoring results of DSTAT and IOSTAT processes on each node. This data is essential for a composite timeline analysis, comparing request times with the resource utilization of the machines.

3. Experiences and results

As a result we get a baseline performance comparison and the idea of problematic characteristics for handling BLOBs in these NoSQL databases, using a conditions related access mechanism.

3.1. Experiences

We faced many unique characteristics of the databases and also difficulties during the testing phase of the research, and we cover them below.

- Limitations

- *Cassandra* - One of the main limitations is that a column's content may not be larger than 2 GByte, however "few MBytes" is a more practical limit. By contrast, the usual size of conditions is "tens of MBytes" which may cause inefficiency in several use cases. However there is a solution, originally implemented in the Astyanax driver[9], that is able to handle this particular problem. We are planning to re-implement this particular recipe in the CMS software framework.
- *MongoDB* - The database supports a unique storage element for large binary files that is called GridFS, that scaled well during the preliminary trial phase. GridFS is a client-side algorithm that splits the large files to small documents and send multiple write requests of sub-documents.
- *RIAK* - This database does not have an object size limit, however practically it aims for small, few MByte values. Either we can implement the splitting solution in the software framework, or we have the possibility to use RIAK CS (Cloud Storage)[10] that is built on top of RIAK and meant to support large file storage in the database.

- Request routing

- *Cassandra* - Controllable policies exist to select a coordinator node for a given request. By using the token-aware policy, the coordinator node can re-route the request to the data holder node to avoid unnecessary data transfer between cluster nodes. With this we have the possibility to ensure that the request is handled by that particular node that stores the requested data.
- *MongoDB* - Sharded clusters have master instances that are aware of cluster topology and about the sharding logic of the collections. So called "*mongos*" instances can connect to the master instances, creating a gateway between the client and the database for request handling. Despite this being a lightweight process, it is not a convenient approach to be integrated in the CMS framework, however it is expected to be easily applicable on the Frontier side.
- *RIAK* - Currently no driver exists for this database that has token-aware routing functionality, and the coordinators are selected with Round-Robin algorithm. This may change with newer versions, therefore we did not renounce on the investigation of this database.

3.2. Results

For each database there are 3 setups $\times$ 2 access types $\times$ 9 throughput levels $\times$ 2 storage types = 108 tests. The maximum throughput, 9 TPS was set on the simple consideration that there is only 1 GBit network available between the auditing engines and the cluster's environment, therefore above 9 TPS we already hit the network limit for the standalone mode. Using a setup with a single database node, the maximum throughput could not reach more than 5 TPS, as after a couple of minutes the number of active users reach a level where we experience request failures. By using the SSD caching layer, we successfully reached 9 TPS even for the single node setups, with an expected steady increase in average request times.

For the cluster setups, we present the results based on our key performance indicator, the average request times compared to throughput. On Fig. 2 and Fig. 3 one can see the average request times for 1 – 9 TPS of the large and medium cluster setups using remote testing without SSD caching. The performance of MongoDB is linear and really good, thanks to the simple fact that the whole dataset was cached in by the database. We can avoid this with a smaller cluster where the dataset cannot fit into memory.

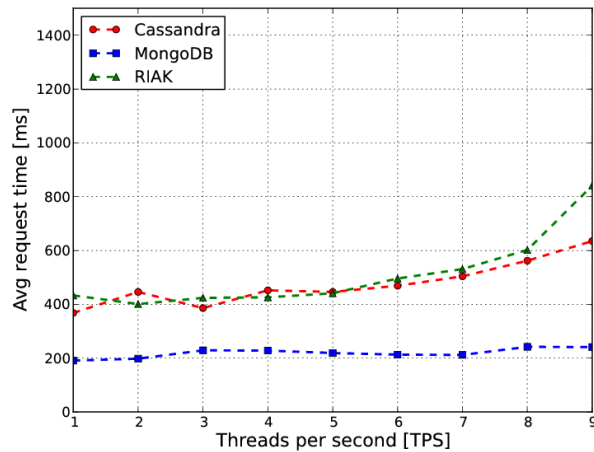


Figure 2. Average request times on large clusters.

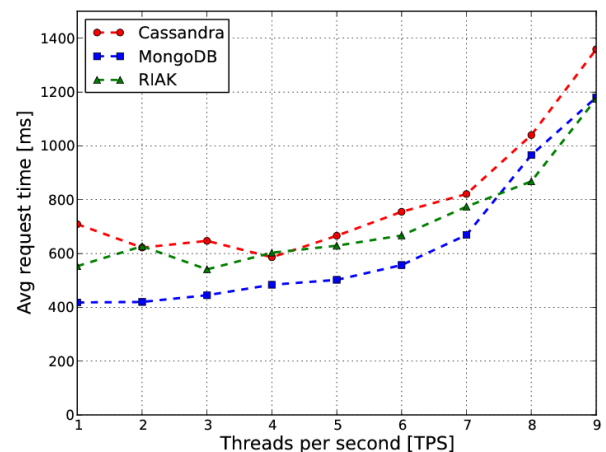


Figure 3. Average request times on medium clusters.

Using the SSD caching on large and medium setups we can see the expected artifact of the "hopping" in the RIAK clusters, and the sudden performance gain for I/O sensitive setups like in the case of Cassandra, shown on Fig. 4 and Fig. 5.

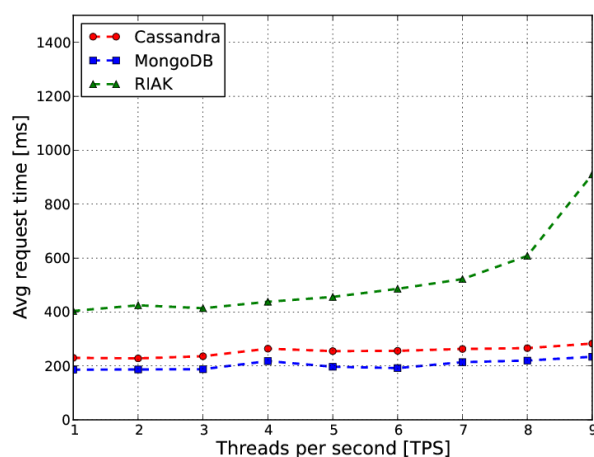


Figure 4. Average request times on large clusters using SSD caching.

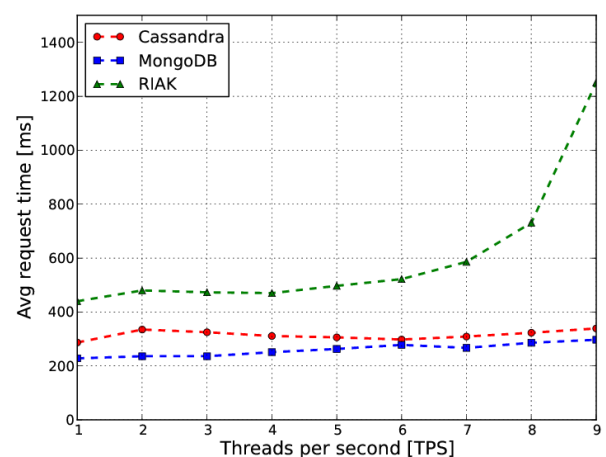


Figure 5. Average request times on medium clusters using SSD caching.

3.3. Ceph

We also tried to avoid the ephemeral storage of the hypervisors and configured the databases to store the data files on a mounted *Ceph*[11] volume. As an example we show a test case with 1 TPS for a single node MongoDB instance in which the data files are stored on an external volume, shown on Fig. 6.

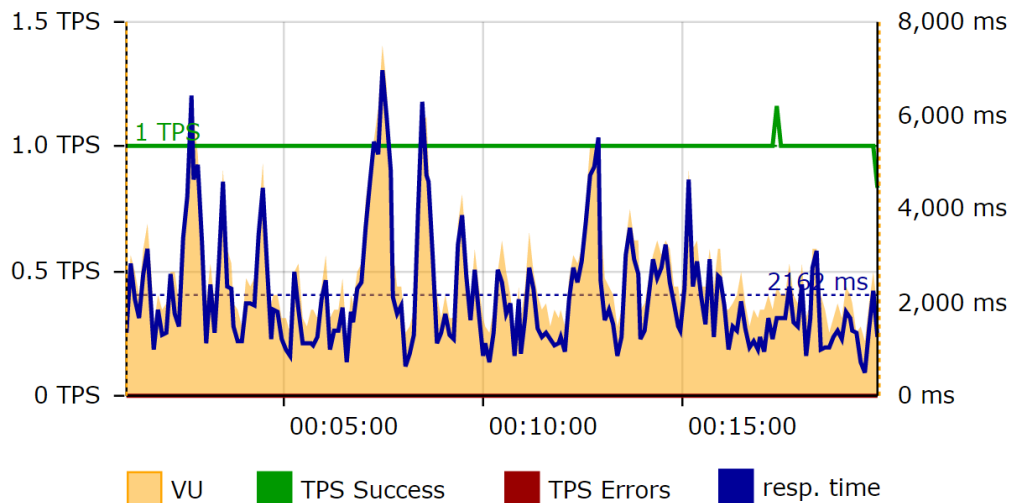


Figure 6. Composite chart for testing Ceph storage for MongoDB.

This particular test resulted with the average request time of 2162 ms and the median of active threads of 2.33. The main reason for this low performance is straightforward, as I/O on a Ceph volume has a latency of $\approx 30 - 50$ ms. As internal database mechanisms frequently do I/O operations, and our read requests divided into several smaller requests as a result of the splitting techniques, the overall latency easily adds up and may reach seconds for single payload request.

3.4. Conclusions

This baseline performance comparison gives a detailed insight about several deployments and how different limitations can affect the performance of these databases. We also showed that with a careful planning a virtual infrastructure can be satisfactory, where the deployed databases can handle high throughput without drastic degradation of the average request time.

We also took into account different storage elements and showed that for our use-case, Ceph volumes should be avoided as the base for the database and SSD cached ephemeral storage has a huge impact on the scalability and performance of any setup.

Based on the study we expect to support stable performance and a highly available state with clusters that are scaled out further on dedicated resources to store real condition data.

4. Summary

We developed a testing infrastructure to measure the performance characteristics and examine the behavior of three well known open-source NoSQL database solutions in the aspects of the Conditions access mechanism. By the observation of these baseline performance results, we were able to elaborate alternative data storing cluster topologies in order to use non-relational data storage for the CMS Conditions. Currently we are working on the integration of different

prototypes based on the observations from these results in order to compare them with the current Oracle solution and also make a formal evaluation of the proposed solutions.

5. Acknowledgments

I would like to express my gratitude to Andreas Pfeiffer for his great help and supervision of the project, to Giacomo Govi for his suggestions and the insight into the current solution. I would like to thank the CMG group for the discussions and last but not least to Arne Wiebalck and the CERN OpenStack team for providing a dedicated environment for the performance tests.

References

- [1] Govi G, Di Guida S, Pfeiffer A, Ojeda M, "The CMS Condition Database system" (2015) this conference. <https://indico.cern.ch/event/304944/session/10/contribution/130>.
- [2] CMS Collaboration, "The CMS experiment at the CERN LHC" (2008) JINST 3 S08004, doi:10.1088/1748-0221/3/08/S08004.
- [3] Pfeiffer A, Govi G, Ojeda M, "Multi-threaded Object Streaming" (2015) this conference. <https://indico.cern.ch/event/304944/session/9/contribution/57>.
- [4] Planet Cassandra, "What is Apache Cassandra", <http://planetcassandra.org/what-is-apache-cassandra/>, [May 14, 2015].
- [5] MongoDB, Inc., "What is MongoDB", Internet: <http://www.mongodb.com/what-is-mongodb>, [May 14, 2015].
- [6] Basho Technologies, Inc., "Relational to Riak", <http://basho.com/assets/RelationaltoRiak.pdf>, [May 14, 2015].
- [7] Apache Software Foundation, "Apache Solr", <https://lucene.apache.org/solr/>, [May 14, 2015].
- [8] Apache Software Foundation, "Apache JMeter", <http://jmeter.apache.org/>, [May 14, 2015].
- [9] Netflix, "Announcing Astyanax", <http://techblog.netflix.com/2012/01/announcing-astyanax.html>, [May 14, 2015].
- [10] Basho Technologies, Inc., "RIAK CS", <http://basho.com/riak-cloud-storage/>, [May 14, 2015].
- [11] Sage A. Weil, "Ceph: Reliable, Scalable, and High-Performance Distributed Storage" (2007) Ph.D. thesis, University of California, Santa Cruz.