A comparison of different database technologies for the CMS AsyncStageOut transfer database

IOP Conf. Series: Journal of Physics: Conf. Series **898** (2017) 042048 doi:10.1088/1742-6596/898/4/042048

A comparison of different database technologies for the CMS AsyncStageOut transfer database

D Ciangottini[4], J Balcas[2], M Mascheroni[5], E A Rupeika[6], E Vaandering[5], H Riahi[8], J M D Silva[1], J M Hernandez[9], S Belforte[3], T T Ivanov[7]

[1]Universidade Estadual Paulista, Sao Paulo, Brazil
[2]California Institute of Technology, Pasadena, CA, USA
[3]INFN Sezione di Trieste, 34127 Trieste, Italy
[4]INFN Sezione di Perugia, 06123 Perugia, Italy
[5]Fermi National Accelerator Laboratory, Batavia, IL, USA
[6]Vilnius University, Lithuania
[7]University of Sofia "St. Kliment Ohridski", Bulgaria
[8]CERN, Geneva, Switzerland
[9]CIEMAT,Madrid, Spain

E-mail: marco.mascheroni@cern.ch

Abstract. AsyncStageOut (ASO) is the component of the CMS distributed data analysis system (CRAB) that manages users transfers in a centrally controlled way using the File Transfer System (FTS3) at CERN. It addresses a major weakness of the previous, decentralized model, namely that the transfer of the user's output data to a single remote site was part of the job execution, resulting in inefficient use of job slots and an unacceptable failure rate.

Currently ASO manages up to 600k files of various sizes per day from more than 500 users per month, spread over more than 100 sites. ASO uses a NoSQL database (CouchDB) as internal bookkeeping and as way to communicate with other CRAB components. Since ASO/CRAB were put in production in 2014, the number of transfers constantly increased up to a point where the pressure to the central CouchDB instance became critical, creating new challenges for the system scalability, performance, and monitoring. This forced a re-engineering of the ASO application to increase its scalability and lowering its operational effort.

In this contribution we present a comparison of the performance of the current NoSQL implementation and a new SQL implementation, and how their different strengths and features influenced the design choices and operational experience. We also discuss other architectural changes introduced in the system to handle the increasing load and latency in delivering output to the user.

1. Introduction

The Compact Muon Solenoid (CMS) experiment at CERN is one of the four major experiments of the Large Hadron Collider (LHC) [1]. It gathers more than 2000 physicists from all over the world. Collected data are stored, processed and analyzed in over 60 data centers on the Worldwide LHC Computing Grid [2]. The data processing activities can be classified into two categories: *organized processing* performed by a central operations team, and *data analysis* performed by individual users. Data analyses, performed with a tool called CRAB[3], produce

output files that have to be transfered to a final *destination storage*, a data center chosen by the user.

Data transfers are an important part of the workflow. Their timely completion is a critical step. Originally, transfers were part of the job execution, i.e., the outputs were copied directly from the worker node producing the data to the destination storage. This model proved to be quite inefficient, causing a significant failure rate and a loss of CPU time in the worker nodes sitting idle while waiting for the transfers to complete. The large number of unmanaged transfers periodically overloaded destination storage sites, particularly those with low bandwidth.

The AsyncStageOut service (ASO) [4] was introduced to solve these issues by centrally managing transfers asynchronously to job execution. ASO has been in production since 2014. Upon job completion, job outputs are copied from the worker node to a temporary area at the same site, called *local storage*. This file transfer happens quickly since the worker node and the storage are close and usually on a local network. The delivery of outputs to the destination storage is then performed in a second stage by ASO. The bookkeeping of those transfers is performed using a central NoSQL CouchDB database [5]. In this paper we discuss why this central database became a bottleneck in the ASO system, necessitating a migration to an SQL Oracle database.

The rest of the paper is organized as follows: section 2 describes the architecture of the AsyncStageOut tool, section 3 elaborates on the interactions between various CRAB/ASO components and the CouchDB database, highlighting major challenges encountered as the number of file transfers increased, and section 4 summarizes strategies explored to address the database scalability problem.

2. Architecture

The ASO-CRAB architecture is shown in figure 1.

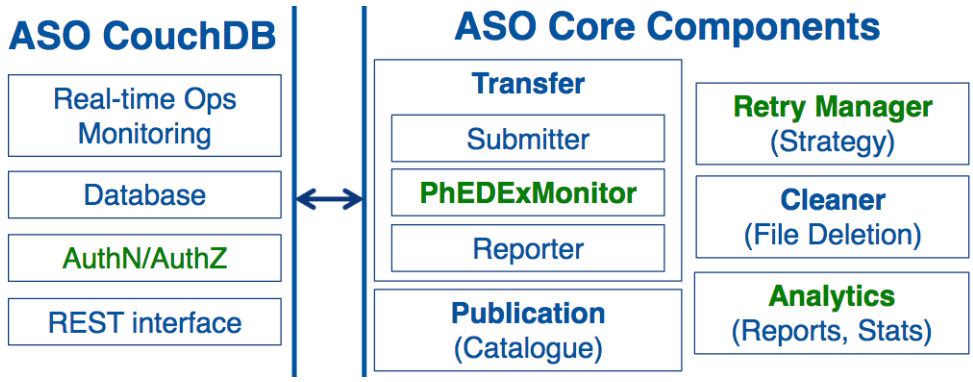

Figure 1. The AsyncStageOut architecture. In green are shown external components used by ASO, in blue ASO components. The ASO CouchDB includes the database itself, a REST interface to access data and a real time monitor used by operators. The core components perform operations on file requests acquired by the database like submission of transfers to FTS, reporting transfer results and cleanup.

The AsyncStageOut architecture consists of two main parts: a series of services that run on a server, called the *ASO backend*, and a central database. This database contains analysis job metadata specifying how files should be transferred from one site to another as well as the transfer state.

An ASO service called the *submitter* daemon periodically reads new database entries and creates a transfer job that is injected into the File Transfer Service (FTS) [6]. The submitter

daemon then updates the database entries, marking the transfer as *acquired*. The *reporter* component monitors the status of transfers, and updates the central database as either *Done* or *Failed*. Another ASO component is responsible for *retry management* of transfer failures. For successful transfers, ASO publishes file information in the CMS dataset bookkeeping system (DBS [7]) and updates the publication state. The design of CouchDB requires creation of a new document for each status update.

After each analysis job, a process called the *postjob* is started on the scheduler. The postjob pushes transfer data needed by ASO, and is responsible for monitoring the transfer state so it can be reported to the user. This polling produces additional requests to the CouchDB.

3. Database scalability

Owing to the many components that need to interact with the database, scalability is a critical feature in the CRAB/ASO system. The number of transfers has continuously increased since ASO was put into production in 2014, and today the load frequently exceeds the initial design requirements of 400k files managed by ASO per day [8] (see figure 2).

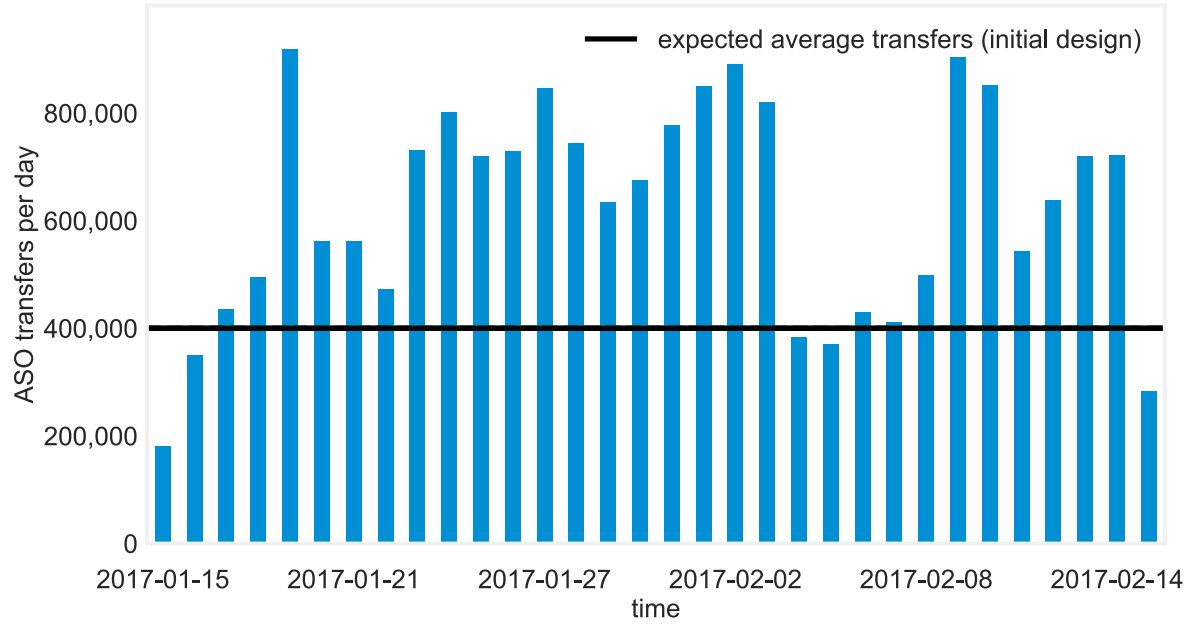

Figure 2. ASO file transfers per day in January and February of 2017. The expected average number of transfers per day in the initial design (400k) is shown as a black horizontal line for reference.

The most critical challenges for ASO are related to the large number of database requests and the frequent document changes. This is exacerbated by the fact that user analysis jobs frequently produce many small files.

The original database technology chosen for ASO was CouchDB. The native REST of the CouchDB interface was flexible and made it easy to use ASO with new clients and workload management systems. The schema-less model of a noSQL database was preferred because it allowed for rapid incorporation of new types of data. However, CouchDB is not well suited for data which change frequently. By design, it creates a copy of the document each time a change happens, and deleted documents are stored internally until a *compaction* is triggered. For ASO that means a new document each time the transfer state changes (as described above). Moreover,

documents are accessed through a set of so-called *views* that cache information. Caches are also based on the principle that old documents are not removed until a view compaction occurs. To purge the database of old replicas and deleted documents, compaction processes run periodically.

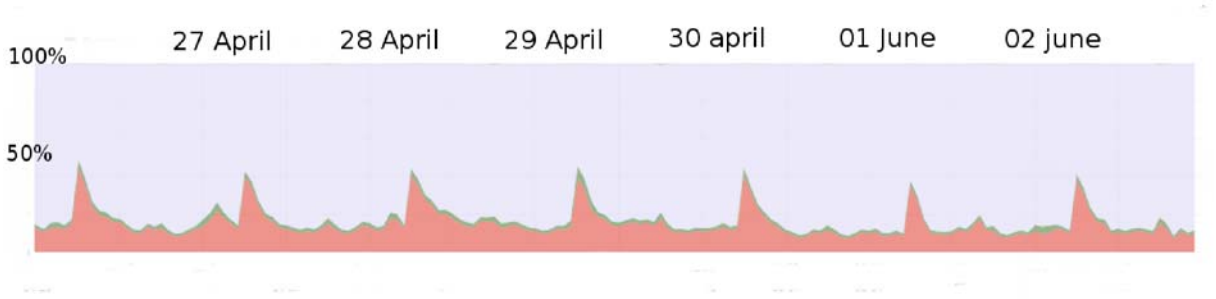

Figure 3. CPU load during compaction time on the machine hosting the CouchDB instance.

For large databases, compaction processes can increase resource consumption on the host machine significantly (see figure 3), which reduces the capacity for normal operations. Moreover, if a burst of requests arrive during that period, they will be processed slowly, resulting in a backlog. This can create a negative feedback loop where a compaction will never complete before the next one is due to start, and the growing cache will rapidly exhaust the available disk space.

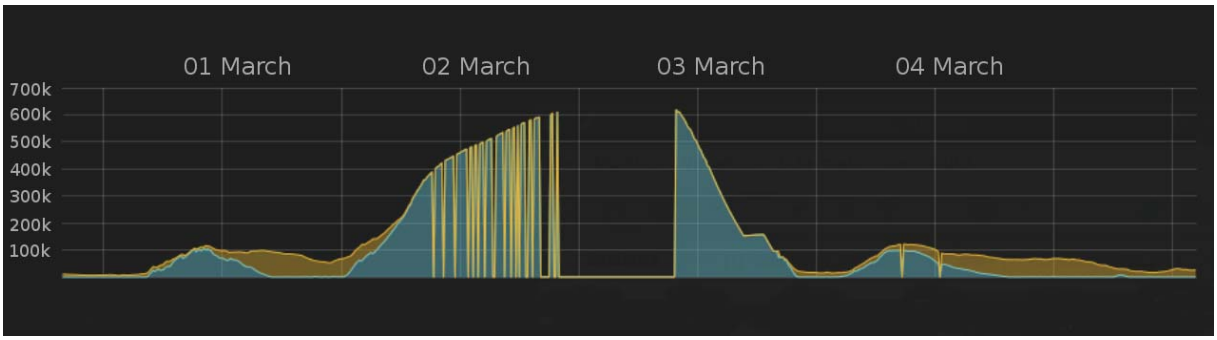

Figure 4. Number of files in states *new* and *acquired* in the CouchDB database

.

figure 4 shows an example of how a peak in new requests is associated with an increase of files in state *new*. This is expected because ASO cannot keep up with processing and CouchDB cannot keep up with indexing the large number of new documents. The indexing of a large number of documents is still running when the view compaction begins. The database response slows down and eventually times out as it is not able to simultaneously compact and keep up with incoming documents. The compaction takes longer than usual under these circumstances and eventually the database size increases out of control due to the backlog of documents in state *new*. At this point recovery is impossible and operators are forced to delete all documents in the backlog to restore the system. This results67 in significant loss of user work.

As ASO approached a constant load of more than 700,000 documents per day and the frequency of incidents such as the one described above increased, it became critical to identify a solution.

4. Explored solutions

To address the scalability challenges, two possibilities were considered: to increase the number of nodes running the Couch database instances, or to switch to another database technology that better suits the use case.

4.1. Horizontal scalability

ASO is structured to be horizontally scalable. The initial deployment consisted of a single CouchDB instance served by an ASO backend. However, CRAB can be configured to partition transfer requests over multiple sets of CouchDB instances and their corresponding ASO backends.

This was the strategy initially adopted to tackle the scalability issue, but it proved inadequate. CouchDB instances have demanding I/O requirements and need special hardware that is not a standard offering of the CERN IT Agile Infrastructure and difficult to obtain. Moreover, adding CouchDB instances and ASO backends proportionally increases the time required by operators to perform maintenance work like periodic rotation of machine databases.

Furthermore, CouchDB has been chosen for historical reasons, but ASO never profited from the map/reduce paradigm typical of CouchDB and other noSQL databases. For these reasons we started exploring other solutions which could decrease the load on CMS operators while increasing the scalability of the system.

4.2. Oracle database

Oracle is one of the most ubiquitous database management systems. Two factors proved to be decisive in the choice of this technology: it is among the services offered by CERN IT, and other pieces of the CMS offline and computing software stack already use Oracle and have demonstrated it to be a robust solution. The fact that it is a CERN IT service lowered the maintenance effort required of CMS operators, and because it is well established in other parts of the software stack meant reduced development effort because knowledge and code could be recycled. This had an additional benefit of increasing homogeneity in the CMS offline and computing software stack.

Required modifications to the ASO backend software, the job wrapper that inserts the transfer request into the database, and other parts of the workload management system (CRAB) have been completed. As shown in figures 5 and 6, early scale tests at production rates look promising.

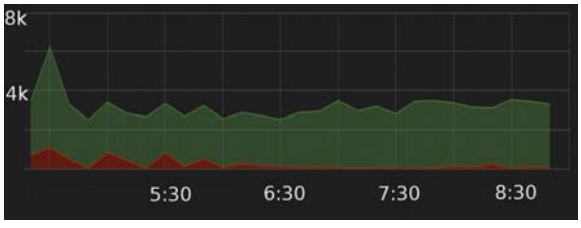

Figure 5. Production rate of 300 jobs / minute.

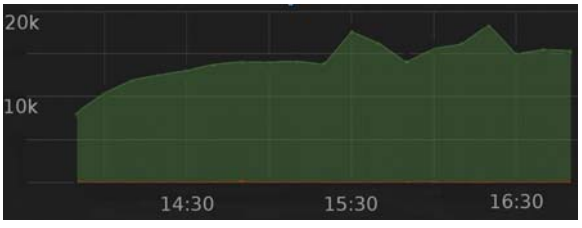

Figure 6. Scale test rate of 1,500 jobs / minute.

As of February 2017 the transfer database is being migrated from couchDB to Oracle, with a small number of jobs reporting to the latter. The plan is to steadily increase the proportion of Oracle jobs until couchDB is completely phased out.

5. Conclusions

In this paper we have presented the improved performance of the ASO component of CRAB achieved by replacing its CouchDB NoSQL database by an Oracle relational database. CouchDB

is a great tool, however, it not best-suited to our use case because data is changing at a high rate, and even the simplest change requires the creation of a new document. New documents needed eventually to be created faster than CouchDB internal compaction and cache refresh (View Build) could deal with them, leading to spiraling up load and even complete system meltdown. Moreover, we never profited from features provided by CouchDB, and the operational issues we encountered led us to move to Oracle, a technology that better suits our needs and is better supported at CERN.

Currently CMS is using both CouchDB and Oracle simultaneously to ensure that functionally all corner cases and potential bugs have been addressed in case they are missed by the integration test suite. The plan is to transparently increase the Oracle share over time until CouchDB is eventually switched off.

References

[1] The CMS Collaboration. The cms experiment at the cern lhc. *Journal of Instrumentation*, 3(08):S08004, 2008.
[2] I Bird et al. LHC computing grid. technical design report. Technical Report CERN-LHCC-2005-024, 2005.
[3] M Cinquilli, D Spiga, C Grandi, J M Hernandez, P Konstantinov, M Mascheroni, H Riahi, and E Vaandering. Crab3: Establishing a new generation of services for distributed analysis at cms. *Journal of Physics: Conference Series*, 396(3):032026, 2012.
[4] M Cinquilli, H Riahi, D Spiga, C Grandi, V Mancinelli, M Mascheroni, F Pepe, and E Vaandering. A glite fts based solution for managing user output in cms. *Journal of Physics: Conference Series*, 396(3):032025, 2012.
[5] Apache couchdb database, http://couchdb.apache.org.
[6] A A Ayllon, M Salichos, M K Simon, and O Keeble. Fts3: New data movement service for wlcg. *Journal of Physics: Conference Series*, 513(3):032081, 2014.
[7] M Giffels, Y Guo, and D Riley. Data bookkeeping service 3: Providing event metadata in cms. *Journal of Physics: Conference Series*, 513(4):042022, 2014.
[8] H Riahi, T Wildish, D D Ciangottini, J M Hernandez, J Andreeva, J Balcas, E Karavakis, M Mascheroni, A J Tanasijczuk, and E W Vaandering. AsyncStageOut: distributed user data management for CMS Analysis. *J. Phys. Conf. Ser.*, 664(6):062052, 2015.