CMS users data management service integration and first experiences with its NoSQL data storage

CHEP 2013

Hassen Riahi (INFN Perugia), Diego Ciangottini (Universitá degli Studi di Perugia/INFN) Daniele Spiga (CERN), Mattia Cinquilli (CERN), Tommaso Boccali (INFN Pisa),
Jose Hernandez (CIEMAT, Madrid, Spain), Preslav Konstantinov (CERN), Valentina Mancinelli (CERN), Marco Mascheroni (CERN), Attilio Santocchia (Universitá degli Studi di Perugia/INFN)

I. Introduction

CMS, the Compact Muon Solenoid experiment located at CERN (Geneva, Switzerland), has defined a model where end-user analysis jobs running on a worker node, where the data reside, store and publish their outputs in the storage element of the user site for later access.
The AsyncStageOut (ASO) is a central service that handles the transfers and publication of users outputs to the required remote storage element. It is designed as a thin application relying only on the NoSQL database (CouchDB) as input and data storage. The highly adaptable design of the ASO has made easy its integration with the CMS/ATLAS Common Analysis Framework (CAF).
We present an overview of the ASO model and the integration strategy with the CAF. The motivations for using the NoSQL technology and the techniques used are presented. We describe the deployment model for the high availability and scalability of the service. We also discuss the hardware requirements and the results achieved as they were determined during the commissioning phase of the Common Analysis Framework.

II. AsyncStageOut model

The direct remote stage-out has been used in CMS since the first versions of the CMS Remote Analysis Builder (CRAB)[1]. This strategy has been demonstrated to work but to have several limitations in the distributed analysis environment. In particular the failure in the remote stage-out is the most common failure mode for analysis jobs. Overall, about 25 to 30% of the analysis jobs fail and about 30 to 50% of the failures are due to remote stageout, so, between 10 and 15% of the cms analysis jobs fail in the remote stage out. Those jobs fail at the end of the processing, so the overall CPU loss is even higher than 10-15%.
To address the synchronous stage-out issues, it was decided to adopt an asynchronous strategy for the remote stage-out. This has required the design and development of a machinery able to stage-out the outputs locally, in a temporary area of the site storage where the code is executed, followed by a subsequent outputs harvesting step where the users outputs are copied to a remote storage using the ASO tool[2][3]. It has allowed to:

- reduce the remote stage-out failure rate affecting the analysis jobs
- avoid the inefficiency of storages due to the unscheduled and potentially concurrent stage-out approach by preventing overloads
- limit the CPU wasting caused by the remote synchronous stage-out of outputs
- automate more the work required by users to manage their files

The ASO tool is implemented as a standalone tool with a modular architecture, based on the common DMWM library, WMCore. It has progressed from a limited prototype to a service managing and monitoring the whole user files steps, namely file transfer and publication. To be as flexible as possible, the ASO tool provides a set of configurable parameters for each of their components. The plugin-based architecture of the ASO allows the interaction with any tool of the Data Management/Workload Management (DMWM) by just writing a dedicated plugin. The data source is set in the configuration of the ASO instance to point to the right plugin. Its core components are described as follows:

- The LFNDuplicator module gets the output details of finished jobs, across the plugin source specified in the configuration file, and stores them as transfer documents in its own database in Couch[4], ASO database.
- The TransferDaemon module instantiates, for each user, a TransferWorker object and submits FTS jobs for transferring the user outputs to the final destination site.
- The DBSPublisher retrieves the files ready for publication from the ASO database and calls a PublisherWorker object for each user to interact with the Dataset Bookkeeping system (DBS) for the files publication. The file metadata required for the publication are retrieved from the cache area specified in the configuration.
- The Analytics component contacts the source across the dedicated plugin to communicate the status of the file management request.
- The FilesCleaner module is tasked to remove, asynchronously, the outputs from the temporary area of the storage source once the file management request is achieved by the ASO.

The ASO Web monitoring is built on top of CouchDB.
To allow the management of many transfer and publication jobs in parallel, the tool implements a parallel processing approach, by means of the multiprocessing library of Python.
Figure 1 shows schematically the ASO architecture and the interactions between their components.

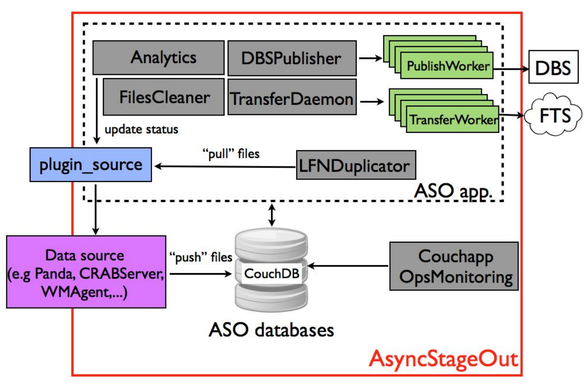

Figure 1: ASO architecture and interactions between their components.

III. Integration strategy

For the distributed analysis tool sustainability, CRAB3 is integrated with the distributed Analysis tool of ATLAS, named PanDA into a Common Analysis Framework. Since the management of the users outputs is mandatory for the execution of the CMS analysis workflow, the ASO has been also integrated into CAF.
A dedicated Plugin, named CMSAdderComponent, has been implemented to be called after the execution of the analysis job. It pushes file documents into the ASO database across the Couch REST interface for the later transfer and publication. In addition, it uploads into a cache area the file metadata required for the publication. Then, the usual ASO machinery described above can start.
To make the ASO communicating with the CAF, a dedicated plugin source has been implemented into ASO. Since the push mode is used for the communication CAF–>AsyncStageOut, this plugin needs just to callback the CAF to update the job status. The messaging technology has been used to achieve that. In particular ActiveMQ is chosen since it is maintained centrally at CERN. The Plugin source makes use of Stomp library to send the callback message including the status of the jobs outputs. The interactions between CAF and ASO are shown in Figure 2.

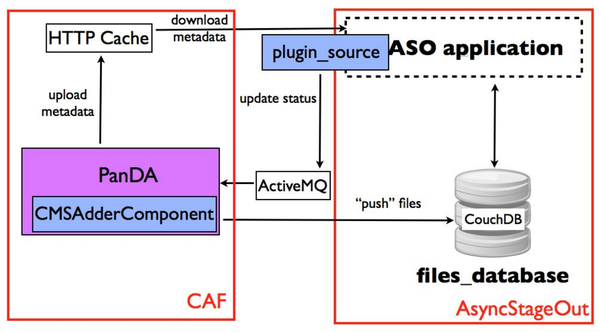

Figure 2: AsyncStageOut and CAF integration

IV. Why do we use NoSQL

The NoSQL database, in particular CouchDB, is extensively used in DMWM of CMS. This technology is used to persist the details of user outputs to manage.
The users data management system is a quite new system for CMS. Its implementation was fast using this technology since no particular database design has been required. Within CouchDB, real-world data is managed as real-world document. In particular, the schema-less of NoSQL models satisfies the need to constantly and rapidly incorporate new types of data to enrich the new applications as the ASO.
Moreover, the REST interface that CouchDB natively exposes has made the tool highly adaptable since it allowed an easy communication with external tools.
The monitoring is an important component for CMS computing group. Indeed, the use of Web databases has facilitated the prototyping and implementation of the ASO Web monitoring. Furthermore, the easy replication and the integrated caching of NoSQL databases, such as CouchDB, influences highly the system scalability and availability.

- Deployment model

Replication synchronizes two copies of the same database, allowing users to have low latency access to data no matter where they are. If one copy of the database is changed, replication will send these changes to the other copy. The continuous replication in CouchDB is a one-off operation. It does not stop after replicating all missing documents from the source to the target and automatically replicates over any new documents as they come into the source to the target. In fact, they are not replicated right away; there is a complex algorithm determining the ideal moment to replicate for maximum performance.
As shown by Figure 3, the replication represents the core of the ASO deployment model to guarantee an high scalability and availability of the service.

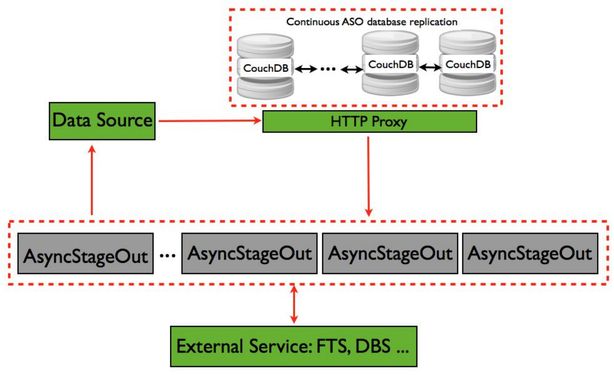

Figure 3: AsyncStageOut deployment model

V. Scalability tests

The ASO foresees the management of nearly 200k of users analysis files per day of close to 1000 individual users per month with minimal delays. A given user can ask also the transfer of the log file which is produced for each analysis job.
Scalability tests are performed to study the ASO response to a high workload independently of the underlying services, namely FTS and DBS. Three different tests have been done. In each of them, the ASO has been loaded with nearly 300k files per day. These tests differs for the number of FTS servers as shown in Table 1. We decided to adopt different strategy of load distribution in order to verify the presence of unexpected issues.
Various scripts have been implemented for these tests, in particular, a transfers injector script to load the ASO and a fakeFTS.pl script in PhedexLifeCycle Agent to simulate the FTS server operations.
Only one instace of ASO has been deployed as well as one istance of CouchDB.

VI. Results

The machine used in these tests is a virtual machine with 8 Cores and 15 GB of RAM. The Operating system used is SLC5.

Test parameters	Test1	Test2	Test2b
FTS server	1	8	8
Daily load	300k files	300k files	300k files
Files/injection	100k	100k	50k
Injection interval	8h	8h	4h

Table 1: Parameters for scalability tests

In Test1 we were mainly interested in validating the workflow for the test procedure. We injected 100k files every 8 hours using only italian destination sites (only one FTS server involved).
Initially, we saw that ASO spends too much time to retrieve the views emitting the value of several parameters. So the views used by the TransferWorker and PublisherWorker components have been split to several ones emitting the value of only one parameters. Furthermore they have been cached for the ASO by retrieving them via crontab jobs.

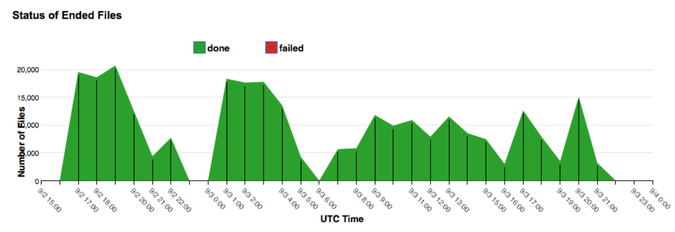

Figure 4: File transferred per hour in Test1

In this test, the first injection of 100k files has started at 16:00. As shown in the Figure 4, the ASO has succeeded the transfer of this load in 7 hours, in particular, before the start of the next injection at 00:00.
After the second injection, the database starts to be slower retrieving views information, maybe due to the number of updates made in the file collection. Probably that is the reason for the second main deep found in the temporal plot in Figure 4. In any case, ASO did not show any issue other than a reduced elaboration speed.
The average elabolaration speed for this test is around 9.6k files per hour with maximum of more than 20k files in an hour.
In Test2 we had two kind of purposes: verifying the scalability with an increased number of FTS server and try to validate hypothesis on load deeps mentioned previously.

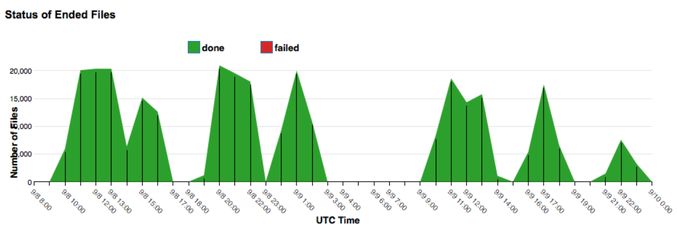

Figure 5: File transferred per hour in Test2

Then we divided this test in two parts, the first one differs from Test1 only for number of FTS server used, results are shown in Figure 5. In this case a wider deep during the second injection is visible and this because of the massive increase of time request for updating data.
In fact, by design, using more destination sites results in more transfer links and so more file collections should be updated.
The average speed of elaboration decreases down here to 8.1k file transferred per hour. However, the Figure 5 shows also that the transfer peaks are more important in this test.

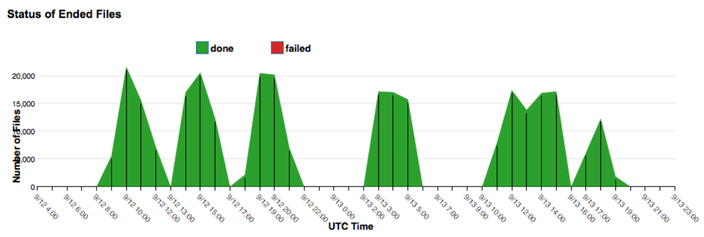

Figure 6: File transferred per hour in Test2b

In the second part, we were interested in studying the injection dependency of the performances, in order to verify if that is the limit of ASO or there is an unrealistic temporal overload in the injection. In Figure 6 a shorter "dead time" can be observed since we injected 50k files every 4h, reaching the same amount of files during a day but distribuited more smoothly in time.
Even though deeps are still evident, the first two are due to absence of new file to transfer until a new injection happens, while wider ones are due to views timeout issue.
The injection strategy in this test, which set is to the half of the load in the previous one, has doubled the number of the wider deep compared to the previous test. In this case the average speed sligthly increases at 8.4k files per hour. Anyway, after those tests, we can assert that ASO appear stable at this load of work even in front of some delays due to file collections update in Couch.

VII. Conclusion

The ASO design based on the NoSQL database, CouchDB, has allowed an easy integration of the tool with CAF. Scale tests of the ASO independently of the underlying services have been performed and have shown satisfactory performances of the tool. In particular there are some issues that have been investigated and, so far, well understood. A summary of the results follows:

- ASO can manage load peak of more than 20k files/h without critical error or crash.
- In an high load scenario a speed lowering in the elaboration has been observed. It mainly depends on the injection distribution strategy.

Actually, the AsyncStageOut can interact only with the Workload management tools such as PanDA. In the future, it will be exposed also to the users. The scalability test infrastructure is ready to be used for the highest load expected.

References

[1] CRAB3: Establishing a new generation of services for distributed analysis at CMS, M Cinquilliet al 2012 J. Phys.: Conf. Ser. 396 032026
[2] Design optimization of the Grid data analysis workflow in CMS, H Riahi 2012 Ph.D Thesis, University of Perugia, Italy
[3] A gLite FTS based solution for managing user output in CMS, M Cinquilli et al 2012 J. Phys.: Conf. Ser. 396 032025
[4] Apache CouchDB database http://couchdb.apache.org.

The work has been partially funded under contract 20108T4XTM of *Programmi di Ricerca Scientifica di Rilevante Interesse Nazionale(Italy)*