A gLite FTS based solution for managing user output in CMS

Mattia Cinquilli
CERN, IT Department, CH-1211 Geneva 23, Switzerland
E-mail: mcinquil@cern.ch

Hassen Riahi
INFN Perugia, Via Alessandro Pascoli, 06123 Perugia, Italy
E-mail: hassen.riahi@pg.infn.it

Daniele Spiga
CERN, IT Department, CH-1211 Geneva 23, Switzerland
E-mail: spiga@cern.ch

Claudio Grandi
INFN Bologna, Viale B. Pichat, 40127 Bologna, Italy
E-mail: claudio.grandi@bo.infn.it

Marco Mascheroni
CERN, Route de Meyrin, 1211 Geneva 23, Switzerland
E-mail: mmascher@cern.ch

Francesco Pepe
INFN Bologna (now at CAEN), Viale B. Pichat, 40127 Bologna, Italy
E-mail: francesco.pepe@bo.infn.it

Eric Vaandering
Fermi National Laboratory, Batavia, Illinois, USA
E-mail: ewv@fnal.gov

Abstract. The CMS distributed data analysis workflow assumes that jobs run in a different location from where their results are finally stored. Typically the user output must be transferred across the network from one site to another, possibly on a different continent or over links not necessarily validated for high bandwidth/high reliability transfer. This step is named *stage-out* and in CMS was originally implemented as a synchronous step of the analysis job execution. However, our experience showed the weakness of this approach both in terms of low total job execution efficiency and failure rates, wasting precious CPU resources. The nature of analysis data makes it inappropriate to use PhEDEx, the core data placement system for CMS. As part of the new generation of CMS Workload Management tools, the Asynchronous Stage-Out system (AsyncStageOut) has been developed to enable third party copy of the user output. The AsyncStageOut component manages glite FTS transfers of data from the temporary store at the site where the job ran to the final location of the data on behalf of that data owner. The tool uses python daemons, built using the WMCore framework, and CouchDB, to manage the queue of work and FTS transfers. CouchDB also provides the platform for a dedicated operations monitoring system. In this paper, we present the motivations of the asynchronous stage-out system. We give an insight into the design and the implementation of key features, describing how it is coupled with the CMS workload management system. Finally, we show the results and the commissioning experience.

1. 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 store their outputs in a storage element for later access. Often resulting files have a significant size (about 1GB per each output file) and the user does not always have his/her own local space resources to store such data. So the outputs are accessed remotely by users for re-analysis. CMS has explored the direct remote stage-out approach: jobs running on the worker node and copying each output file to a user pre-defined location at the end of the job execution. This is also called the synchronous approach. Recently, a more evolved approach to the stage-out step has been implemented: once jobs have produced their output files, these are copied to the local site storage element, close to where the jobs have run. A central service is then able to schedule the transfers to the required remote storage elements. With reference to the strategy of the stage-out step execution, this approach is called asynchronous stage-out. This paper describes the asynchronous stage-out solution presenting the motivations and the details of the AsyncStageOut service, the test and results performed with this tool, and gains compared to the synchronous solution.

2. Experience with current synchronous stage-out.

The direct remote stage-out has been used in CMS since the first versions of the CMS Remote Analysis Builder (CRAB). This strategy has been demonstrated to work but to have several limitations in the distributed analysis environment. Figure 1 illustrates the combination of the jobs and input/output data workflows. The success of output files transfer from a local disk to a remote storage element relies on the fact that network, storage resources and all infrastructure elements in the middle behave without any problem. Within this approach, the job wrapper will retry the operation again a short time after failure. Obviously this is not efficient to occupy the CPU if copying the output takes a large fraction of time compared to the data processing. In general, in case of an infrastructure problem, it is unlikely that it can be resolved in a short interval of time. If each attempt fails, the job wrapper continues its execution and clears the output of that job, effectively losing it. The job must then be resubmitted and the resources used by the last job are wasted.

The failure of the stage-out may be due to network problems, the data storage system, or authentication problems on the sites involved in the transfer of the output. The failure of the stage-out is the cause of inefficiencies since it not only occurs after the use of computing resources required to execute the job, including the queues of Grid and local schedulers, but also because the resubmission of the job is responsible for a delay in the completion of the task, forcing users to wait a longer time before being able to analyze their outputs.

For the CMS Tier-2s, this strategy can often result in a Distributed Deny of Service attack (DDoS) to the Grid sites' storage systems where a huge number of worker nodes dispersed widely

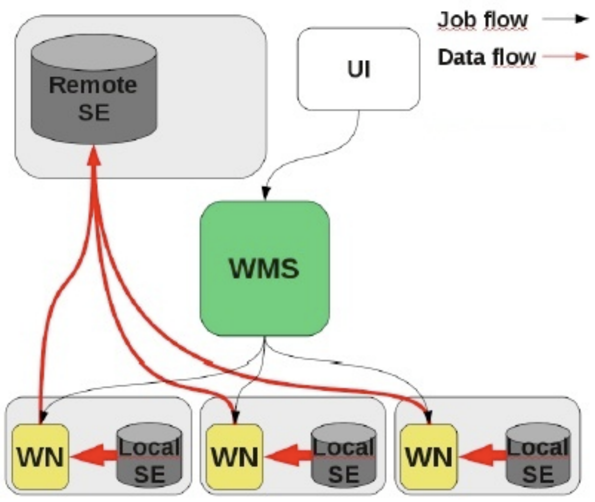

Figure 1. Jobs and data workflows. The jobs are submitted to the WMS which redirects them to the matching sites. Jobs run in the site's worker nodes reading the input data to analyze from the storage element. Once the job has succeeded and produced output files, these are then copied by the job itself to the pre-defined remote storage element.

are trying to stage-out there.

During July 2011, a crucial month for physics conferences, 4 766 902 analysis jobs have been submitted, where 4 067 588 has finished successfully and 14 391 have failed in the remote stage-out (numbers taken from Experiment Dashboard [2]). Such a defect in the design of the analysis workflow often causes the failure of the users' analysis jobs after reaching the timeout of the Grid copy command. Figure 2 shows in (a) the fraction of the jobs failed in 2010 and, among the jobs failed for application error, (b) shows the percentage of failure caused by the remote stage-out approach. The distribution of the number of successful jobs vs. CPU wall-clock time spent for the remote stage-out is shown in Figure 3. The 95% of successful jobs spent an average of 500 seconds while performing the remote stage-out which means that they consumed about 22 000 days of CPU wall-clock time. This represents the latency introduced by the remote stage-out step in the physics analysis activity. The total amount of CPU wall-clock time consumed doing the remote stage-out from the worker node corresponds to about 24500 days (Figure 2 (b)).

Regarding failed jobs, Figure 4 shows the distribution of the number of jobs that failed in the remote stage-out vs. the average CPU wall-clock time spent and the distribution of the same number of failed jobs vs. the average CPU wall-clock time spent for the remote stage-out step. The manual or automatic resubmission of a failed job can happen only after the end of the execution of the current job. Figures 4 (a) and (b) also shows that about 4600 days of CPU wall-clock time were effectively wasted by 14 391 jobs. In addition, the collaboration had to wait at least that time before the resubmission of their jobs. Figure 4 shows in (b) that the remote stage-out has consumed about 330 days of CPU wall-clock time, which also correspond to the delay in case of resubmissions.

3. Asynchronous stage-out strategy and workflow

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 the storage of the site where the code is executing, followed by

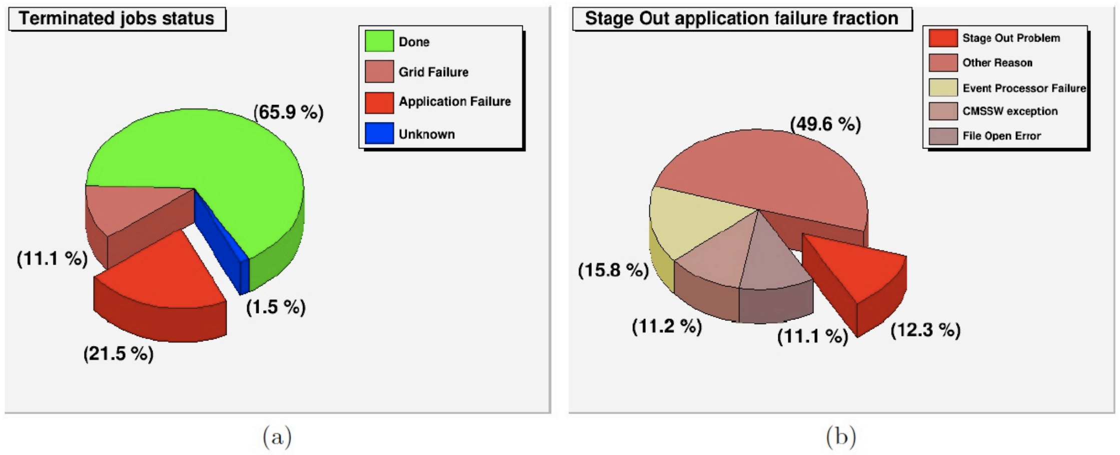

Figure 2. Fractions of CMS analysis jobs grouped by terminated status in (a) and *Application Failure* error types in (b), highlighting the percentage of failures caused by the synchronous remote stage-out; statistics from 2010.

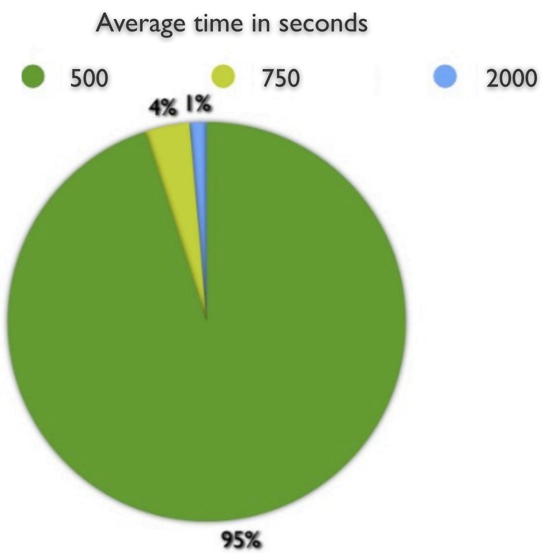

Figure 3. The distribution of the number of jobs done during July 2011 over average CPU wall-clock time spent for the remote stage-out.

a subsequent outputs harvesting step where the users outputs are copied to a remote storage using gLite FTS [3].

The asynchronous stage-out workflow is shown schematically in Figure 6. The use of gLite FTS helps balancing site resource usage, prevent network and storage overload, and manage transfer resubmission. The latter will avoid wasting resources by resubmitting the whole analysis workflow as it is the case for the synchronous remote stage-out approach and reduces also the delay in the execution of the analysis workflows since only the transfer of the output will be resubmitted. To reduce the delays in accessing the outputs, the initial copy of the output, in the local SE of the site where the job run, is provided to the user. The analysis workflow including the asynchronous stage-out steps can be summarized as follows:

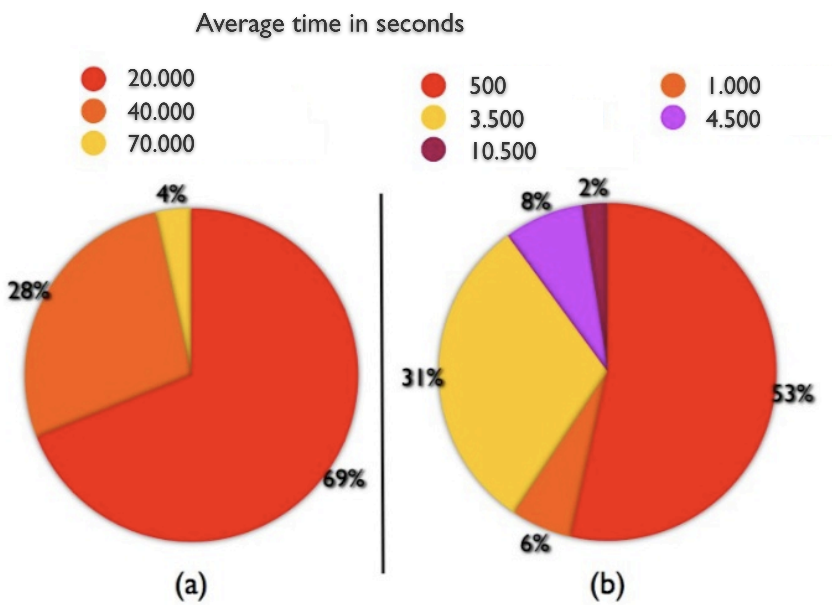

Figure 4. (a) The distribution of the number of jobs failed for the remote stage-out during July 2011 over CPU wall-clock time spent. (b) The distribution of the number of jobs failed for the remote stage-out during July 2011 over CPU wall-clock time spent by the remote stage-out step.

- user submits its analysis workflow from its working station (User Interface),
- based on the configuration and the location of the data that he needs, the jobs are scheduled to run in matched Grid sites,
- once the execution of the analysis code in the worker node (WN) is done, the output is copied in the local SE of the site (local stage-out) in a /store/temp/user logical area; if the local copy of the output succeeds, a request is automatically submitted to FTS to copy the output to the remote SE (remote stage-out) in /store/user area,
- the transfer request is then tracked and resubmitted if the submission fails.

The NoSQL database, CouchDB [4], is extensively used in Data Management / Workload Management (DMWM) [5]. This technology is used to persist the details of user outputs to transfer using the AsyncStageOut tool[7].

3.1. AsyncStageOut design

The asynchronous stage-out tool is implemented as a standalone tool with a modular architecture, based on the common DMWM library, WMCore [6]. Figure 5 shows the sequence of the interactions of AsyncStageOut in CRAB3 architecture. To be as flexible as possible, the AsyncStageOut tool provides a set of configurable parameters such as expiration days, max transfers retries, max files per transfer and database source. Its core machinery is described in the following items.

(i) AsyncStageOut interacts with the database source in CRABServer's CouchDB instance to get the details of the output to transfer; each document in this database represents a job FrameWorkJobReport (FWJR) and should be organized in steps describing the status of the analysis job execution in the WN, namely logArchive, cmsRun and stageOut. The analysis job output is copied locally in the storage of the site where the code is executing.

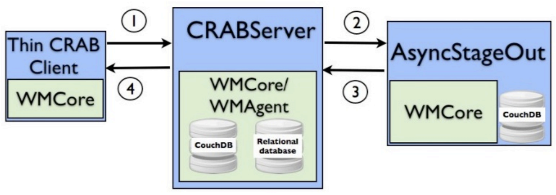

Figure 5. CRAB3 architecture overview including AsyncStageOut service.

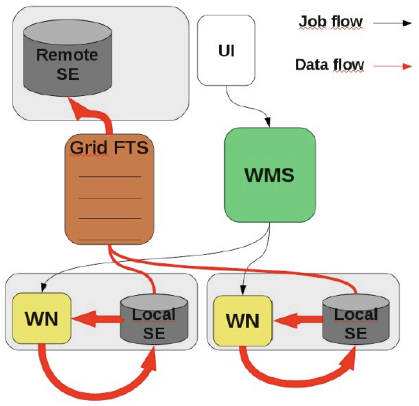

Figure 6. Asynchronous stage-out workflow schema.

Once this step is finished, the analysis job is considered as done and its information, such as status, output location and output path are updated in its FWJR.

(ii) The LFNDuplicator module regularly polls the database source to get the output details of finished jobs and stores them as transfer documents in a CouchDB database, named files_database. Once added, the transfer document is in a *new* state.

(iii) The TransferDaemon module regularly polls the files database to get the details of files to transfer. At each polling cycle it instantiates, for each user, a TransferWorker object which marks the transfer documents as acquired, and then submits FTS jobs for transferring the user outputs to the final destination site and:
  - if a transfer fails, the transfer document is marked as failed in the files database; the TransferWorker marks the transfer as expired if the transfer still fails after `max_transfers` retries times;
  - it marks the transfer as done if it succeeds.

(iv) Updates the database source by adding an AsyncStageOut step to the FWJR document.

The main interactions of the asynchronous stage-out components are shown in Figure 7. To allow

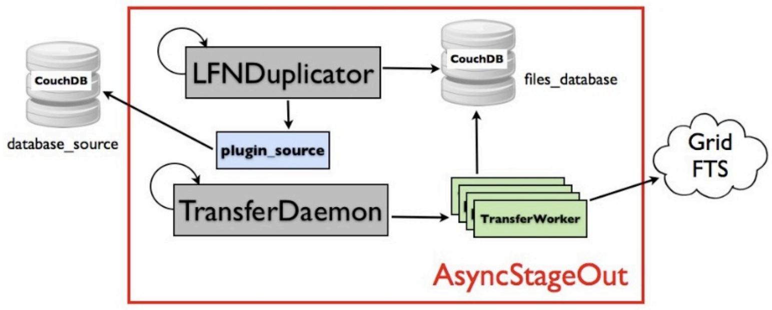

Figure 7. Asynchronous stage-out core schema.

the management of many transfer jobs in parallel, the tool implements a parallel processing approach, by means of the multiprocessing library of Python. Expired and done transfer documents will remain the in files database for `expiration_days` before being removed from there by the StatDaemon which tasked to merge the transfer documents, by FTServer used, into a single document and insert it into the stat database.

To be able to interact with any DMWM tools requiring the remote stage-out service, this tool is developed following a plugin-based architecture to allow addition of a new source whenever needed by just writing a class which inherits from the Source parent class. The first plugin developed is the Job State Machine (JSM) which interacts with the WMAgent framework. Another plug-in has been recently developed which will use summary documents produced by all agents and replicated to a central monitoring database.

The asynchronous stage-out system requires much more care, in particular regarding the use of disk storage and time required by the operation of the remote stage-out. In fact, in such a system, each output must be copied first in the local SE and transferred later to the remote site, which requires more storage capacity than the current synchronous remote stage-out strategy. In addition, the operation of the local copy and the subsequent transfer with FTS, introduces latency between the completion of the job and the time at which the asynchronous stage-out tool begins effectively to transfer the output. For this reason, the asynchronous stage-out system has been integrated into a version of CRAB2, and used to perform preliminary tests of scalability and latency.

4. Results

Since CRAB3 is not yet in production and there was the need to test the asynchronous stage-out with real use cases. The first functional tests have been done plugging the AsyncStageOut service into CRAB2. This required an ad-hoc version of CRAB, that was able to bypass the remote stage-out and instead store the output locally. Also the AsyncStageOut had to interface with CRAB2 system, so a new plug-in had to be developed in order to achieve that.

4.1. Functional tests

Tests used a dataset spread over 20 available CMS sites which resulted in a to create a large number of jobs producing equal output sizes. The CRAB workflow created for this test consisted of 400 jobs, each producing a 1.3 GB file. The workflow was submitted several times to reach a total of 1686 transfer jobs. Figure 8 shows the fraction of successful and failed transfers, while Figure 9 shows the failure rate by site.

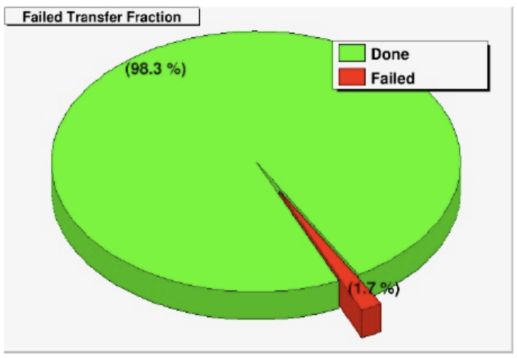

Figure 8. Fraction of failed and succeeded transfers for functional tests.

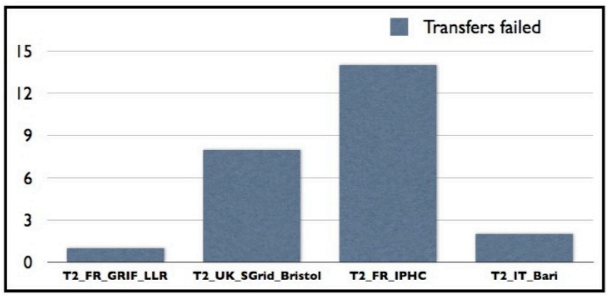

Figure 9. Transfers failed by site for functional tests.

Out of a total of 1686 transfers, approximately 98.3% of transfers succeeded. The transfer failure in this test were caused by using an incorrect FTS server or temporary issues at these sites involved in the transfer. The configuration of the asynchronous stage-out allows to retries failed transfers. Given that, the asynchronous system can be considered to be more reliable than synchronous one. The histogram in Figure 10 shows the number of transfer attempts by site. This histogram shows the results only for the successful jobs. Note that if the system was synchronous, the only successful jobs would be the ones in the first column, while the others would collapse and their jobs would have to be resubmitted. From Figure 9, the number of attempts seems to be greater for some sites. As example, T2_UK_Bristol appears to be one of the sites with the highest number of failures (Fig. 9): this might indicate some incompatibility of the site with the asynchronous stage-out system, since the latter is still experimental. The

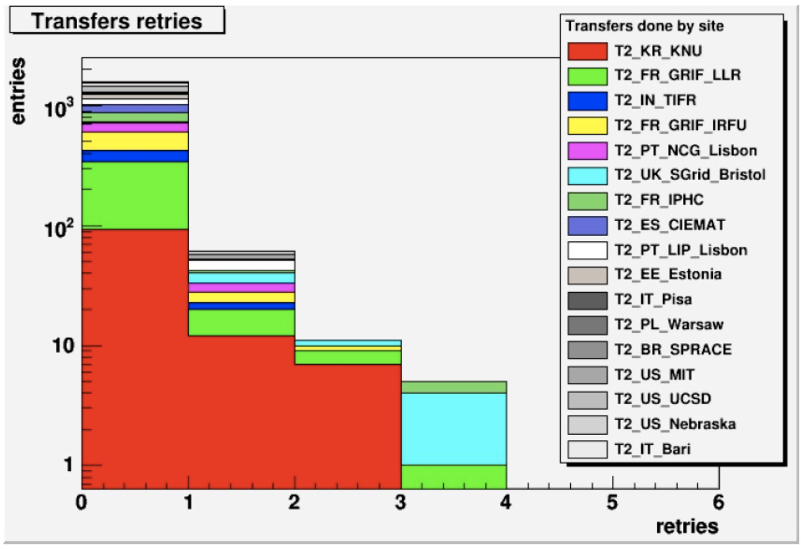

Figure 10. Asynchronous transfer retries by sites.

total latency introduced by the asynchronous stage-out system is more distributed in the second round of tests, due to differing network performance at the various sites and the different times when the transfers of outputs were requested. Furthermore, the output sizes of each job were very similar, nearly 1.3 GB. Table 1 shows the average time required to transfer an output from a site source to a site destination. The performance was varying between sites.

The load supported by the asynchronous stage-out server during these tests is shown in Figure 11. The histogram shows the trend of the total number of transfers per hour, and the number of transfers marked as completed to the asynchronous stage-out server for each hour. The maximum number of simultaneous transfers reached almost 600 jobs. Within the conditions described above, the system did not show scalability problems. The maximum number of transfers marked as completed per hour was approximately 350. Comparing with Figure 11, the number of analysis jobs completed per hour in the first half of December 2010, a period in which there was intense analysis, the number of jobs completed is over two orders of magnitude greater than the results obtained in our test, shown by the green histogram in Figure 12. However it is not realistic to test with loads comparable to those of real analysis, because the consumption of resources would be very high.

Table 1. Transfer latency by source site of about 1.3 GB file.

Source site	Latency (min)	Files
T2_US_UCSD	158	14
T2_EE_Estonia	312	86
T2_FR_GRIF_LLR	610	254
T2_FR_GRIF_IRFU	381	183
T2_IN_TIFR	390	92
T2_US_Nebraska	6	109
T2_PT_LIP_Lisbon	540	142
T2_ES_CIEMAT	318	137
T2_KR_KNU	400	113
T2_FR_IPHC	425	149
T2_PT_NCG_Lisbon	352	126

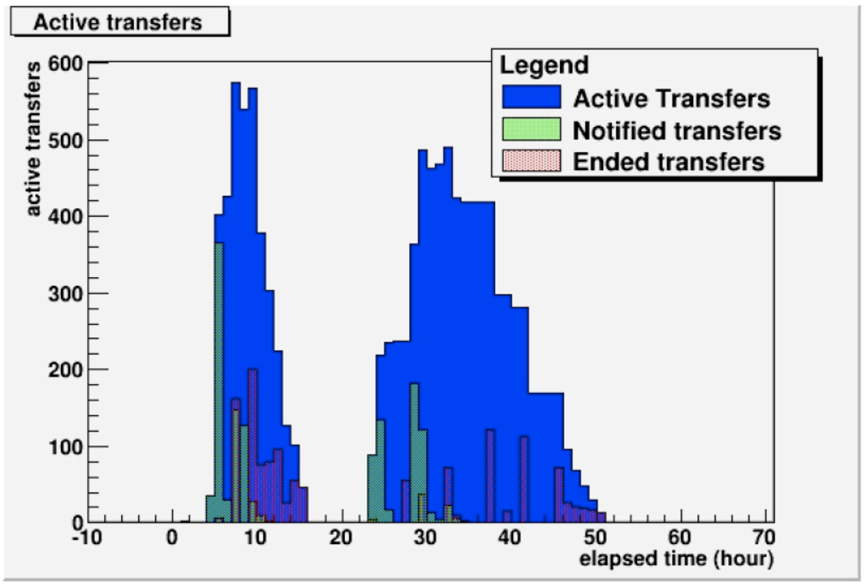

Figure 11. Load supported by the asynchronous stage-out server to manage 1349 transfers in 60 hours. In blue, the number of active transfers for each hour. In green, the number of transfers notified in the asynchronous stage-out for each hour, while the red represents the number of transfers completed in each hour.

4.2. Scalability tests

More tests have been done by using the new generation of analysis tool, CRAB3, in order to understand the behavior and possible bottlenecks of the asynchronous solution. The total number of jobs submitted in this test is 23 909 jobs, where 8 414 successfully to executed their codes and staged-out the outputs locally while 14 495 failed. Figure 13 shows the success and failure rates by user of the asynchronous transfers of outputs. Most of the failures of jobs in these tests were caused mainly by the fact that at the time of testing CRAB3 was still a prototype and sites were not yet compatible with CRAB3 workflows.

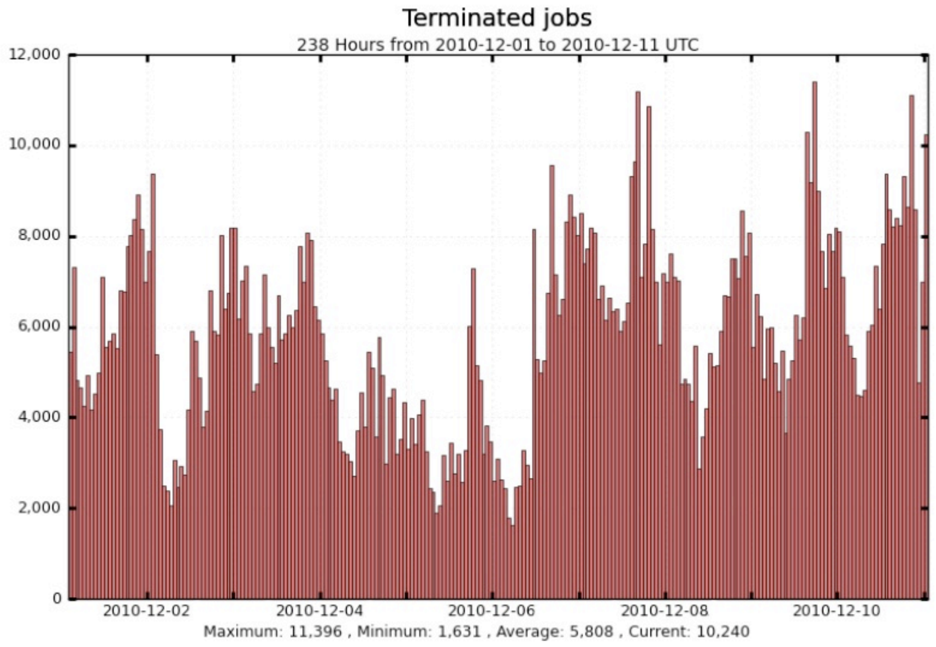

Figure 12. Number of analysis jobs terminated per hour during the first half of December 2010 (the period when these tests are done).

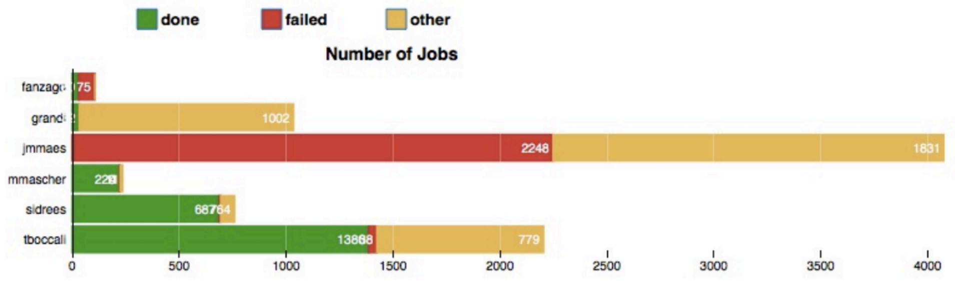

Figure 13. Success and failure rates by users.

5. Conclusions

Based on Figure 2, if the jobs were submitted using CRAB2 and, so, the remote stage-out step is performed synchronously, an approximation of the number of failed jobs for the stage-out from a total of 8 414 would be 223 and the number of succeeded jobs would be 5 545. So, 223 CPU slots would be effectively wasted. Moreover, as shown by the plot (b) in Figure 4, failed jobs make the physicists or CRAB resubmission components wait that time before the resubmission of the jobs.

Regarding succeeded jobs, the asynchronous stage-out approach makes a first safe copy available to users in `/store/temp/user` area in the storage of the site where the job was executing. So, physicists can already access their outputs from there and in a transparent way through the CRAB3 RESTFull API's [?].

The distribution of the number of jobs finished vs. average time spent for the remote stage-out in CRAB 2 is shown in Figure 4, while the distribution of the number of jobs done during these tests vs. average time spent for the local stage-out is shown in Figure 14. If the jobs of this test were submitted using CRAB2, 95% of jobs done, which is 5 268 jobs, would spend an average of 500 seconds to perform the stage-out, while they required only an average of seconds to perform the local stage-out in CRAB3. So, in this test, the asynchronous stage-out approach has allowed to users the access to their outputs about 450 seconds earlier for 5 268 jobs compared to the synchronous stage-out. This approximation of the gain in term of CPUs slots and times seems to be promising.

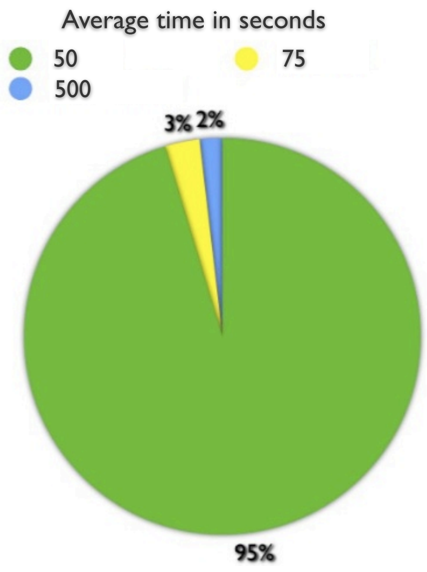

Figure 14. The distribution of the number of jobs done during these tests over average time spent for the local stage-out.

The new stage-out approach implemented by the AsyncStageOut system will allow CMS analysis jobs to:

- regain CPU times previously consumed by the analysis job stuck in the remote stage-out step;
- reduce the delays in completing the analysis workflows by removing the fraction of failures caused by the synchronous stage-out step;
- regain the resources wasted previously by the failure of jobs to perform the remote stage-out synchronously;
- organize the transfers to Tier-2s sites using a dedicated service in order to avoid the overload of their networks and storage systems;
- reduce the latency of the availability of analysis jobs outputs by storing them first in `/store/temp/user` area of the sites where the jobs were executing.

The AsyncStageOut has been heavily tested by the integration team of CMS and has shown to have satisfactory performance. Once CRAB3 is in production it will be possible to better understand the system behavior, scalability and whether, in particular, it is introducing issues to PhEDEx transfers of CMS production system by overloading FTS servers, designing and implementing then eventually required optimizations.

References

[1] CMS Collaboration, Adolphi, R., et al.: The CMS experiment at the CERN LHC, JINST, 0803, S08004 (2008).

[2] Karavakis, E., et al: User-centric monitoring of the analysis and production activities within the ATLAS and CMS Virtual Organizations using the Experiment Dashboard system, in proceedings of EGI Community Forum 2012 - EMI Second Technical Conference, 2012, PoS(EGICF12-EMITC2)110.

[3] Frohner A, et al. Data management in EGEE. Journal of Physics Conference Series 219 062012, doi: 10.1088/1742-6596/219/6/062012, 2009.

[4] Apache CouchDB database `http://couchdb.apache.org`.

[5] Kutzentsov, V. et al.: Life in extra dimensions of database world or penetration of NoSQL in HEP community, in these proceedings.

[6] Wakefield S, et al. Large Scale Job Management and Experience in Recent Data Challenges within the LHC CMS experiment. Proceedings of XII Advanced Computing and Analysis Techniques in Physics Research, 2008.

[7] Riahi H., 2012: Design optimization of the Grid data analysis workflow in CMS, Ph.D. Thesis. University of Perugia, Italy.

[8] Cinquilli M., et al.: CRAB3: Establishing a new generation of services for distributed analysis at CMS, in these proceedings.