Processing of the WLCG job monitoring data using ElasticSearch

September 2015

Author:
Javier Delgado Fernández

Supervisor(s):
Edward Karavakis
Julia Andreeva

CERN openlab Summer Student Report 2015

Abstract

The Worldwide LHC Computing Grid (WLCG) includes more than 170 grid and cloud computing centres in 40 countries. More than 2 million computational jobs are being executed on a daily basis and petabytes of data are transferred between sites. Monitoring the job processing activity of the LHC experiments, over such a huge heterogeneous infrastructure, is really demanding in terms of computation, performance and reliability. Furthermore, the generated job monitoring flow is constantly increasing, which represents another challenge for the monitoring systems.

While existing solutions are traditionally based on Oracle for data storage and processing, recent developments in the SDC monitoring team evaluate different NoSQL solutions for processing large-scale monitoring datasets. Among those solutions is ElasticSearch – an open source distributed real time search and analytics engine. The aim of this project is to prototype the WLCG Job Monitoring applications to store and retrieve data using ElasticSearch.

Table of Contents

Acknowledgments

My sincere thanks to my supervisor, Edward Karavakis who spent a lot of time helping me in each detail of my project, verifying the correctness of all my tasks, motivating myself in every moment and taking into account all my suggestions or ideas. But I would like to say him thanks also for all the help he gave me in a lot of topics out of my job.

Thanks also to my office mates because they welcomed me and took me into their group of people in CERN. And also to Pablo because of his help with some parts of the project and for being so attentive.

And finally, I am very grateful to my section (specially to Julia) for make me feel part of their team.

1 Introduction

The purpose of this project consist on the improvement of the persistence system in terms of performance. To achieve this goal, the new implementation relies the data on other paradigm of persistence based on documents (JSON) instead of relational schemas.

1.1 Experiment Dashboard

The Experiment Dashboard[1] system is a python framework that provides powerful tools allowing us to know a lot of information about the data transfer, the job processing and the status of the distributes sites and services of the WLCG. The system is heavily used by the LHC experiments in order to follow job and data transfer execution, detect and investigate inefficiencies and failures, assisting in commissioning new sites and services and identifying trends in the WLCG. The solutions that the Experiment Dashboard offers are used by different categories of LHC users; from physicists running their jobs on the Grid and user support teams to site administrators, shifters and LHC management.

These solutions are generic and were designed to be used in several LHC experiments, keeping as much code as possible shared in order to reduce the maintenance cost and the effort for the development of new features.

1.2 WLCG job monitoring

The Worldwide LHC Computing Grid is a collaboration between several countries around the world providing computational resources required in the LHC environment. The reason to build this complex structure was the huge amount of data that LHC needs to deal. It is estimated that LHC experiments are currently generating 25 PB[2] which should be accessed by 7000 physicists.

Taking into account this large amount of data that should be stored, analysed and processed by the WLCG infrastructure in more than 180 computing centres with different configurations, we should realize that there are a lot of different kind of issues that can occur. For this reason, we need to monitor all the processes that are being executed along the time, providing us a good knowledge about the performance and reliability of this enormous infrastructure.

The monitoring of the job processing activity is one of the several services that Experiment Dashboard is offering. It is providing a complete picture of the jobs processing status over the WLCG. Currently, the grid infrastructure receives more than 2.0 million of jobs per day to be processed, and this number continues to grow.

ATLAS and CMS are the current users of the job monitoring provided by Experiment Dashboard and have different ways to submit and analyse jobs. In the case of ATLAS, they are using a centralized Workload Management System called PanDA[3] and for the CMS use-case, they are using a system called CRAB3 to submit analysis jobs and WMAgent to submit production jobs. The Experiment Dashboard Job Monitoring applications are sharing the same database schema in Oracle and the same Web-based User Interfaces. For the purpose of this report, we will only cover the ATLAS job monitoring activity.

2 ElasticSearch

ElasticSearch[5] is a new system designed to provide all the components required in order to add search capabilities to software projects. ElasticSearch is build on the top of Lucene and is using Java as a platform. It offers a RESTful interface to query data and uses the JSON format not only for the documents but also to query the data contained on it.

Internally, this technology stores the JSON documents that we insert and also creates an index for every field that it has. This behaviour produces a really low latency queries even when the data to analyse is enormous.

Due to this, ElasticSearch is an ideal candidate when you need to deal with huge amounts of time-series data and you can adapt your project to work with this kind of NoSQL documental database. Particularly, they are focusing their attention in real time data, multitenacy and text search, bundled with the advantages of NoSQL document databases such as schema less, distributed computing and storage, high availability and fault tolerant.

In the case of the current project, ElasticSearch is used as a NoSQL document database and it is not used to perform queries on free text.

Some studies were conducted before the start of this project[4] that were testing the performance of several technologies with the Experiment Dashboard monitoring data showing that ElasticSearch is the best approach to our current problem in terms of performance.

3 Project Overview

When you need to deal with such amount of data, like in our current problem, and return the result within a few seconds, you are being prompted to research other ways to solve it. A regular query on this system, for example, to show the landing page, consist on several aggregations of one million records with one hundred fields per each record.

3.1 Objectives

This project has as a main objective the performance improvement of ATLAS job monitoring in the Experiment Dashboard.

3.1.1 Reduce the latency of dashboard job monitoring web page.

Although the job monitoring web page of dashboard has a good responsiveness, it was obtained thanks to the use of other complementary techniques such as distributed caching with memcached. This project aims to improve the performance of the queries executed against the storage system, making the user interface more responsive for its users.

3.1.2 Increase the time-range limitation on the queries

Due to the improvements in the terms of performance, now you can execute other queries with much more data getting the results in a reasonable time. So ultimately, the job monitoring web application can offer aggregation of big ranges of dates.

3.1.3 Reduce the dependency of external tools

Making use of less external tools reduces the huge effort required to maintain all the modules. Another benefit is the reduction of problems in our system originating from third parties software.

3.2 Architecture

The code evolving this project is developed mainly in Python but also in ElasticSearch DSL. It is based in a lot of independent applications but sharing a lot of common code. Each one of these applications is called a dashboard agent or "collector" and has a well defined interface based in a constructor with the parameters specified in an XML configuration file and a method called "run" where the collector executes its work.

3.2.1 Collectors involved in importing ATLAS Jobs

To enable ElasticSearch as a new persistence system instead of the previous solution with Oracle, we need to complete the information of a job that is caused by the lack of information in the primary information source when we are parsing a job. So the collector needs to look for additional data in other parts of the system.

The main collector in our solution is called "Panda", which is reading the data from ATLAS PanDA system and parsing it to store in ElasticSearch, but this collector can not do its work properly because it requires to decode some fields that are incomplete. To show an example, we receive from Panda the ATLAS PanDA[6] queue name of a site and not the official WLCG site name, but when we are using the Dashboard, we want to know the official name of the site. For this reason, we first need to import the association between the ATLAS PanDA queue names and the WLCG sites using the ATLAS Grid Information System (AGIS) and then to perform the translation which is executed when the job is parsed in order to achieve a great performance.

These helper collectors are:

- AppGenericStatus: Used to decode the error exit-code originating from the application, the specific reason of failure for a job and a more general category of error reasons.
- GenericStatus: Used to decode the error exit-code coming from different grid and ATLAS services. Just like the AppGenericStatus, it includes the error exit-code, the specific reason of failure for a job and a more general category of failures.
- PatternAGIS: Used to retrieve the association between ATLAS PanDA queues and official WLCG site names using the AGIS system.
- PatternOracle: Used to import historical association entries between ATLAS PanDA queues and official WLCG site names by querying the appropriate table in the current production persistency solution implemented in Oracle.
- GenericType: Used to import the task types (i.e. analysis, production, test) by querying the appropriate table in Oracle.
- SitesResolver: Used to import site names, tiers, countries and WLCG Federations by querying the appropriate table in Oracle.
- ProdSimulation: Querying the production system of ATLAS (ProdSys2) to retrieve metadata for the simulation type of production jobs, i.e. Fast Simulation or Full Simulation.

These previous side-collectors are needed to allow the Panda collector to work accurately. But panda collector is aided by some other related collectors, like an archive collector that parses the historical data with the same behaviour of PandaCollector or PandaUnparsedJob that is trying to

reprocess the jobs that were not parsed in previous attempts due to incomplete information for a specific job. The last collector, called "Menu" is used to quickly retrieve all the unique values that can be found in the filters view of the real-time job monitoring user interface. It is reading all the jobs and looking for all the unique values in order to show the options you have available in each filter field.

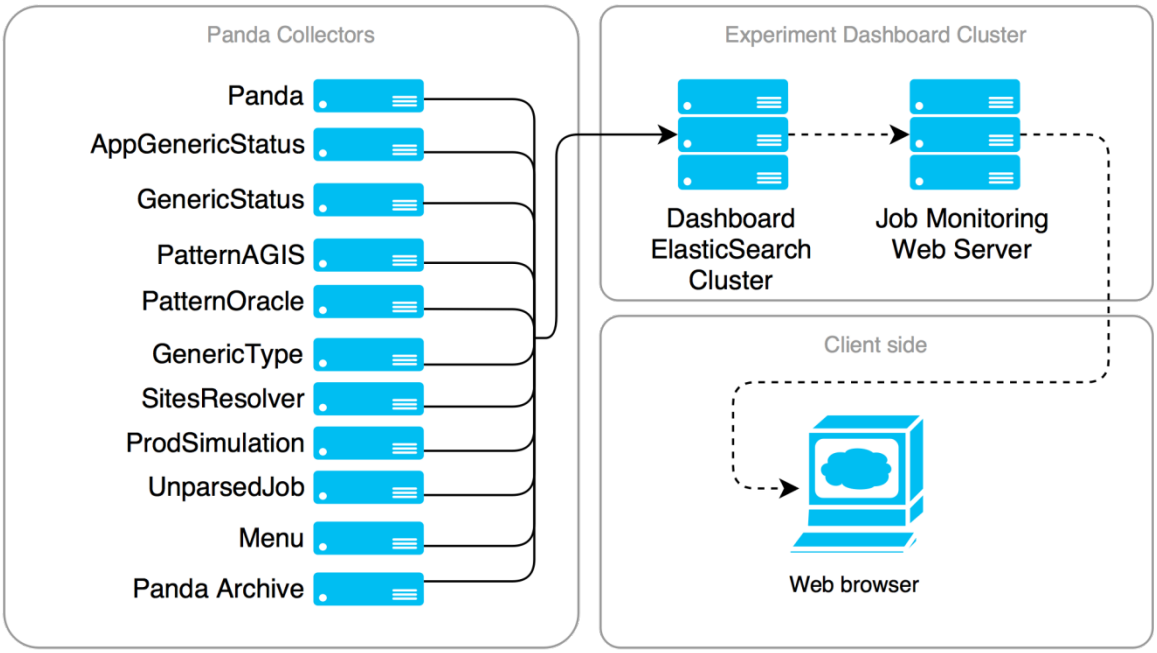

Figure 1 Panda collectors disposition.

3.2.2 Job import workflow

The job collector has been designed to parse jobs in a fault tolerant way ensuring that it will keep on working even under unexpected conditions.

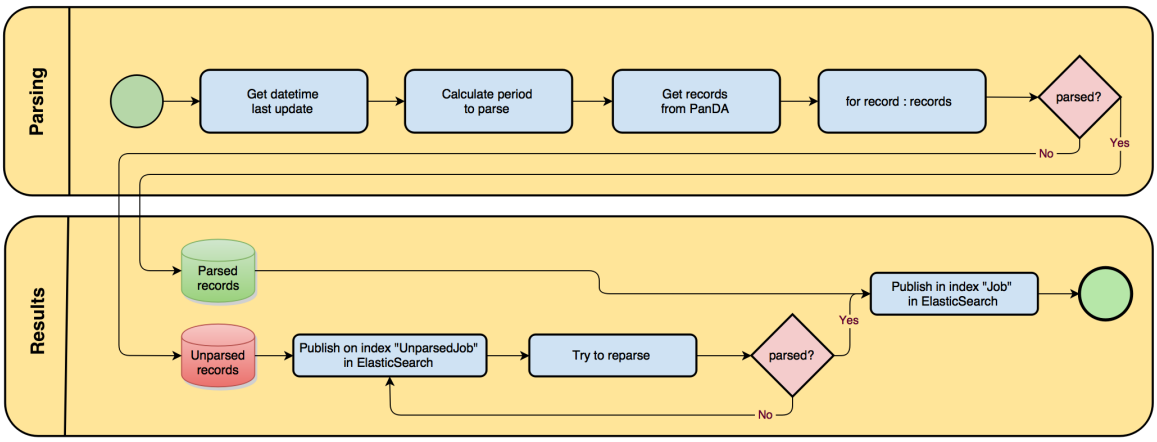

Figure 2 Job collector workflow.

In case of one of the steps showed in the above figure fails, the process will start again from the beginning with the same parameters of the previous execution. Depending on the error type, the system could restart the complete parsing or throw the package to a special storage with all the

relevant information regarding the insertion/parsing problem. The project also provides a system that monitors this special storage for any failed insertion attempts. It will then try to re-parse them as the failure might have been caused due to some missing information during the first collector attempt that could have been inserted by now.

Due to the fact that the persistence system is too different from the previous implementation, this project also provides another collector that works with historical PanDA data, with the same behaviour of the real-time panda collector.

4 Project Results

After the execution of this project, we have reduced the number of the technologies required in job monitoring module of dashboard. Furthermore, we are keeping more consistency in terms of languages and technologies that are being used in this module.

4.1 Job collector improvement

The new job collector is developed in Python instead of Python and PL/SQL as the previous module. And in terms of performance, the new collector gets a speed up of 2.6x the performance of the previous implementation using a PL/SQL procedure to translate the PanDA raw data into the common Experiment Dashboard Job Monitoring schema.

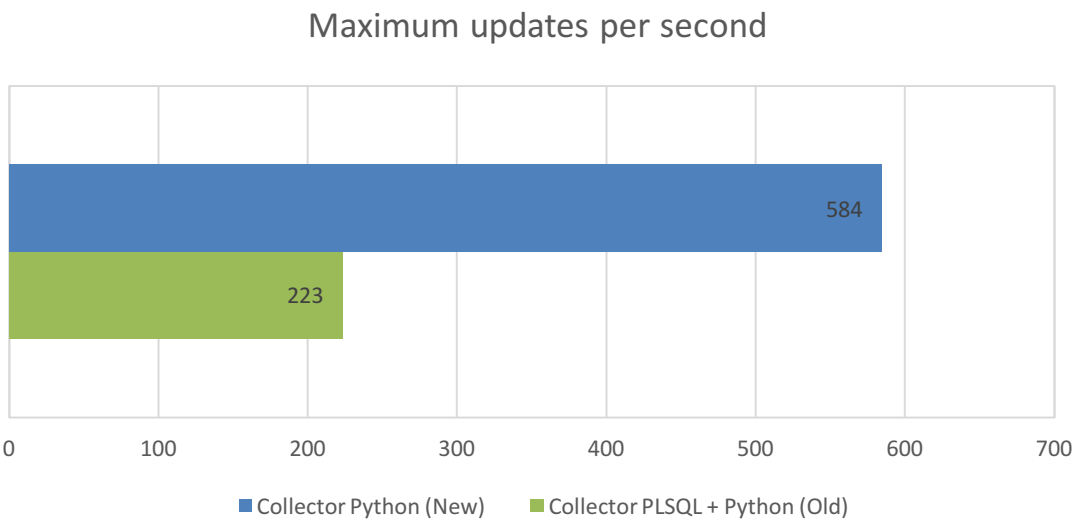

Figure 3 Performance of job collector.

This improvement will aid to fix some problems occurred due to peak values. On average, the new collector will deal with 180 updates per second but for times that the data received is higher than expected, the new collector will have more leeway.

The new collector has been tested importing successfully all ATLAS jobs from the beginning of 2015 until the end of August which corresponds to more than 175 million job records.

4.2 Modified User Interface

The project also required to modify the User Interface in order to provide access to ElasticSearch (the new persistence system) instead of Oracle. This task consists of modifying the parts of the data access object previously developed to work with Oracle to use ElasticSearch by generating the queries on the fly, taking into account all the selected filters and the sorting by attributes.

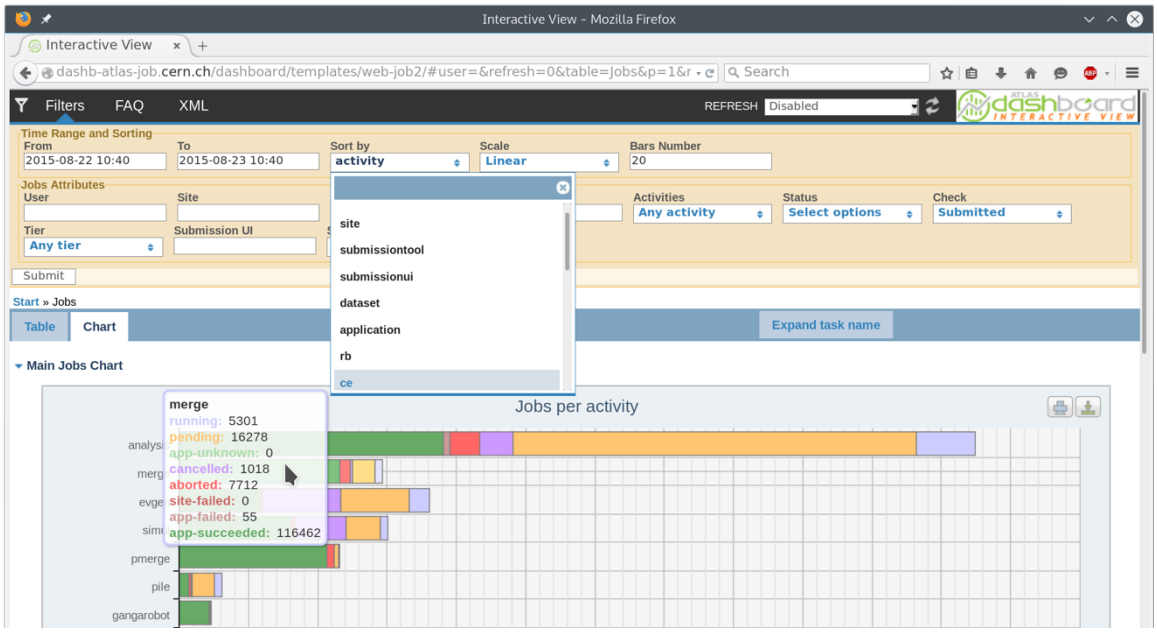

Figure 4 Screenshot of the current user interface.

The picture above shows the current view of the main application of this project, which is dynamically generated on the client side. All the filter fields have autocomplete features using the possible values from the data stored.

4.3 Dashboard responsiveness

With the new implementation of this persistence system, we are getting a user interface that is more responsiveness, more dynamic and more interactive. The previous implementation also was limiting the possibilities due to the time of execution. This is for example the case of queries with a range of more than one month.

When we analysed the performance, we have encountered a real improvement in the application. The first benchmark revealed that the response time of the landing page has been greatly improved as Figure 5 shows. The new implementation performs the query in less than one second and taking into account that the previous implementation was using around 12 seconds we can talk about noticeable improvement.

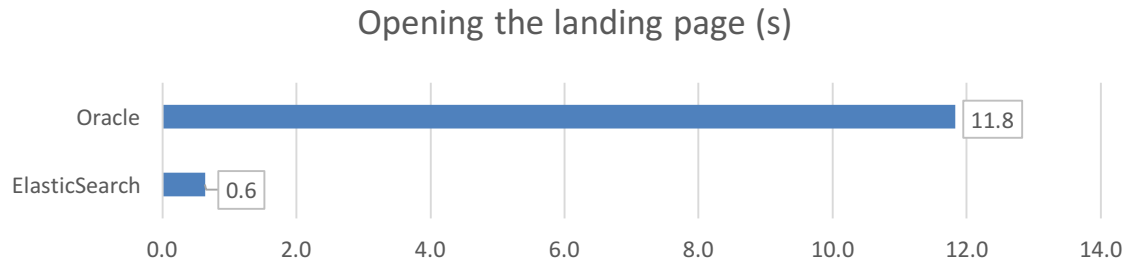

Figure 5 Opening the landing page chart.

But, a query of historical data will convince us totally that this new implementation is greatly improving the system. If we need to retrieve information about 6 months ago, the previous persistence system, will spend more than 700 seconds as we can see in Figure 6, but ElasticSearch is only spending 0.8 seconds.

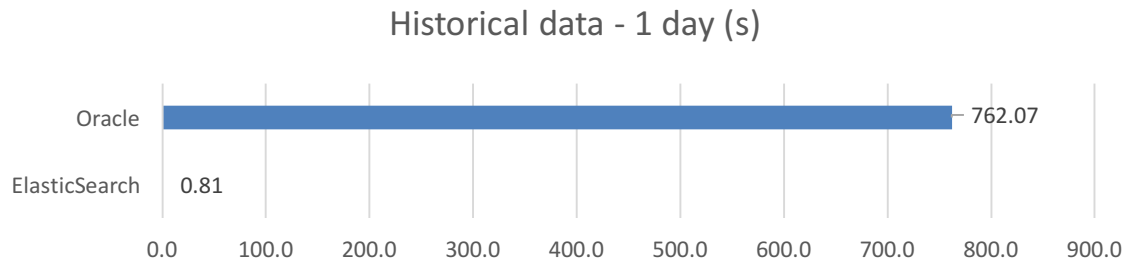

Figure 6 Query data from 6 month ago.

And finally, when we query large term periods, we have again the same results, the previous implementation can not be used to this purpose meanwhile the new implementation using ElasticSearch keeps offering a good response times as we can see in Figure 7.

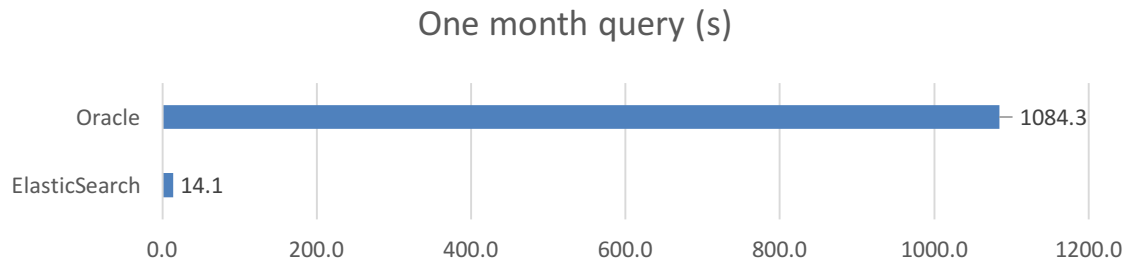

Figure 7 Time spent in a query with one month of data.

5 Future work

As in every project, there are always a list of tasks to be done. In the current project, we have four big tasks to be completed in the long term to get the system fully migrated and in production:

5.1 Extend the system to job accounting

The next task to do related to this project should be the extension to the job accounting system. This other system is aggregating individual job records together and creates accounting summaries allowing the users to view job accounting data over long time periods. Actually, this system that creates aggregated job summaries will use the new real-time job monitoring system that was implemented in this project.

5.2 Fix some parsing problems

Testing our parsing system, we have encountered around 2e-5% of jobs with problems during its parsing. It is not a representative number but the parser should be improved in order to achieve a proper parsing of as many records as possible.

5.3 Port the CMS collectors to store data in ElasticSearch and the User Interfaces to retrieve data from ElasticSearch

Since the ElasticSearch schema is common between ATLAS and CMS, a future work project would be to also port the CMS collectors to store data in ElasticSearch and to modify the User Interfaces to query data from ElasticSearch. This project has shown that ATLAS users will benefit a lot from using the User Interfaces that are running on top of ElasticSearch. It would be nice to repeat this exercise for CMS as well so that their users will benefit from the great speed improvements.

6 Conclusions

As a result of this project, we can conclude that ElasticSearch has a great performance, even dealing with huge amounts of data and this study has shown that by using ElasticSearch in the ATLAS Job Monitoring project, the speed of the system has been greatly improved. This speed improvement will also benefit the individual ATLAS users using the real-time Experiment Dashboard User Interface.

Also is noteworthy to mention the easy adaptation of classic workflows with relational databases into the schema of a documental NoSQL system like ElasticSearch.

By using ElasticSearch, an open source project, the costs can be reduced as there are no licencing costs. Also, it is easy to handle extra load by scaling-up the system by adding more servers in the cluster without having to pay for any licence.

7 References

1. Andreeva J et al, “Experiment Dashboard - a generic, scalable solution for monitoring of the LHC computing activities, distributed sites and services”, 2012 J. Phys.: Conf. Ser. 396 032093 doi:10.1088/1742-6596/396/3/032093
2. http://wlcg-public.web.cern.ch/about
3. Andreeva J et al, “ATLAS job monitoring in the Dashboard Framework”, 2012 J. Phys.: Conf. Ser. 396 032094 doi:10.1088/1742-6596/396/3/032094
4. J Andreeva et al, “Processing of the WLCG monitoring data using NoSQL”, 2014 J. Phys.: Conf. Ser. 513 032048 doi:10.1088/1742-6596/513/3/032048
5. https://www.elastic.co/products/elasticsearch
6. E Karavakis et al, “Common Accounting System for Monitoring the ATLAS Distributed Computing Resources”, 2014 J. Phys.: Conf. Ser. 513 062024 doi:10.1088/1742-6596/513/6/062024