Modernizing the Monitoring of Mass Storage systems

September 2016

Author:
Rosy Gupta

Supervisor(s):
Hervé Rousseau
Sebastien Ponce

CERN Openlab Summer Student Report 2016

CERN openlab

Project Specification

The work is a project of IT-ST group, dealing with mass storage systems at CERN and responsible for more that 150PB of data stored on disks and tapes. The task consists of modernizing the logging system of one of of the storage systems namely CASTOR, in particular by integrating it into common monitoring framework provided by the IT department and by moving to new display technologies. The main tools used are Hadoop/HBase, Spark, Python, and web technologies like Django, Graphite and Grafana.

Abstract

The LHC has entered a new regime by producing more than 25PB of data since the beginning of this year. Thus the storage and monitoring of log messages is of crucial importance for the researchers. The CERN Advanced STORage system (CASTOR) is a storage system developed at CERN and is used by LHC experiments to store physics data files. Monitoring of this complicated distributed system is mandatory in order to assure it's stable operation and improve its future performance. This project presents an infrastructure meant to collect and display monitoring information about the system by the means of Graphite and Grafana data visualization platform.

Acknowledgement

A sincere thanks to my supervisor, Hervé Rousseau who spent time in helping me with each detail of the project, verifying the accuracy of all the tasks, motivating me at every step and taking into account all my suggestions or ideas . I am grateful to have been involved in such a project and learning experience. Sincere thanks to Sebastien Ponce for his help now and then with the project matter and the overall project supervision. The project would not have been possible without the work and cooperation of my colleague Alexandre Terrien who worked simultaneously as a summer student with me. Thanks also to my office mate, Michal Simon for helping me understand the CERN IT infrastructure and new technologies. Finally, I am very grateful to my section (FDO) and the group (IT-ST) for making me feel a part of their team.

Table of Contents

Introduction

The contribution is a part of the renovation of the infrastructure of the main CERN storage system, CASTOR as it manages the bulk stream of data from LHC and other experiments. To efficiently manage the service, a solid monitoring infrastructure is required, able to understand and analyse the log messages. The system developed and deployed acts as a monitoring plugin to the simultaneously newly developed streaming agile infrastructure and leverages on technologies such as Django, Graphite and Grafana. This section deals with the existing infrastructure as well as the motivation behind the project revealing the problems faced.

Background

CASTOR

The CERN Advanced STORage manager (CASTOR) is a hierarchical storage system which was developed at CERN to serve LHC experiments data storage requirements. It consists of a high performance disk cache tier, a slower but reliable massive storage tape tier and several daemons, which are responsible for catering to user requests and managing file transfers. This multi-layered architecture is masked from the end-user as CASTOR provides a logical, UNIX like namespace.[cas]

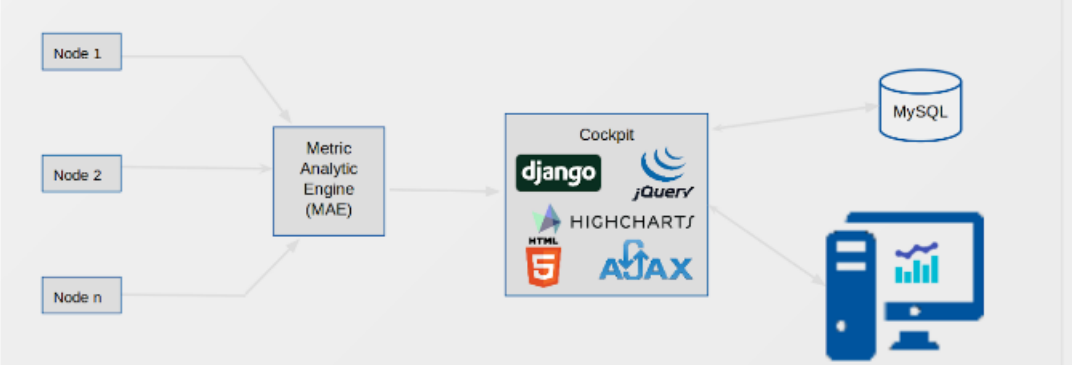

Figure 1: Existing infrastructure

Django

Django is a web application development framework written in Python that aims to handle many of the tedious tasks related to web development, such as URL parsing, session management, database queries and so on. It is based on a model-template-view architecture, where a model is a class with fields mapped to columns in a relational database, the template is the HTML code with support for a special template language used to insert data from views, and the view is the component that retrieves data from the model, and serves it to the template, possibly with additional logic added. The models and views are both written in Python.[dja]

Metric Analytic Engine

The Metric Analytic Engine project is a ”framework” for metric computation developed at CERN.It can be applied to any type of key-value messages.[MAE] During the project, MAE evolved into MAE-Spark where the computation ran on spark. Apache Spark is a fast and general processing

engine compatible with Hadoop data. It can run on Hadoop clusters through YARN or Spark's standalone mode, and it can process data in hadoop Distributed File Storage. It is designed to perform batch processing.

Cockpit

Cockpit is the web interface, developed at CERN, for displaying the data computed by MAE. It displays charts with metric data and stores the data into the local DB.MAE pushes the list of all its running metrics i.e Python dictionaries via HTTP POST request, with available data. Thus the Cockpit can synchronise its running metrics. The cockpit also talks with the MAE using RPyC server to fetch information about the metrics. Cockpit is built on Django web framework and heavily uses HTML5, CSS3 and AJAX/Javascript. The graphs are built using static configuration and the state of the graph is in HTML. The Javascript is used for the user-interface only and processes the options to draw the graphs with Highcharts. The Javascript to draw the graph is handled by a JQuery Plugin. All the information needed to draw the graph is fetched from the server via AJAX POST requests.[coc]

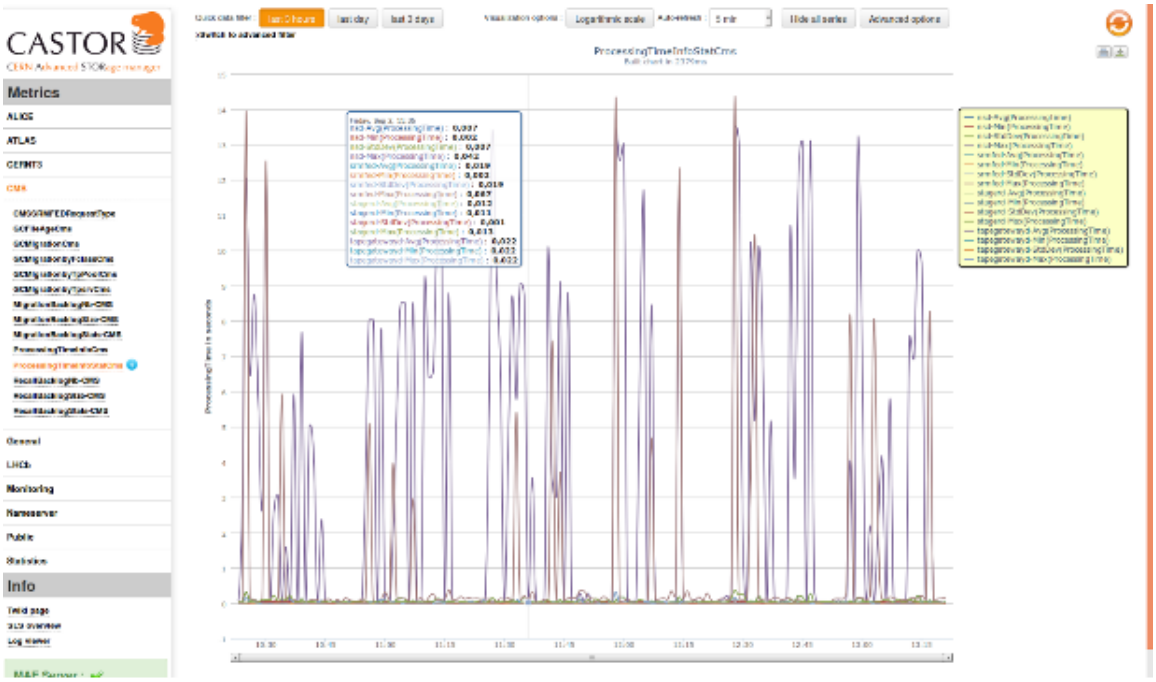

Figure 2: CASTOR Cockpit dashboard

Motivation

The current monitoring infrastructure does not live upto the standards of processing and storing such huge amounts of data efficiently.

1. Metric data quick growth

   Currently in CASTOR's monitoring stack, data points are stored every 60 seconds without any aggregation. The historical metric data is not of much use with such high granularity for longer term trend monitoring. This means the metric data consumes a lot of disk space for

long histories. At present, the volume is as high as 90 million data points approaching a file size of 140PB on disk.

2. Latency

   Another problem with CASTOR's monitoring infrastructure is the latency for displaying metrics in the Cockpit dashboard. Loading and plotting just 3 days of data points takes around 25 seconds.This delay is not optimal as a lot of significant events can happen while a remarkable number of machines are being monitored. Also, the current framework has an extremely verbose codebase.

3. Lack of time-series database support

   The time based indices are costly to maintain for the current local db of Cockpit i.e MySQL. Because you want to query the data on a time range the schema is required to be indexed on time. Since the index is always growing and outgrows memory eventually becoming a challenge for querying. Also, the current system cannot really fetch data from big time ranges.

Literature Review

In the following section of the document, some background information about the new framework services that had to be incorporated is given.

Graphite

Graphite does two main functions: It stores numeric time-series data and renders graphs of this data on demand. It consists of 3 software components:

1. carbon - a twisted daemon that listens for time-series data. It stores the metrics temporarily in a memory buffer-cache for a brief period before pushing them to disk in the form of the Whisper database format.

2. whisper - a simple database library for storing time-series data which is similar in design to RRD.

3. graphite webapp - A Django webapp that renders graphs on-demand using Cairo.

Graphite uses a multiple retention scheme which down samples metrics (averaging by default) as threshold retention are crossed. So you can store long histories of metrics while saving on disk space.

```
Retention = 15s:7d, 1m:21d, 15m:15y
```

This implies that the data will be stored at a frequency of 15sec for 7 days and as soon as the threshold frequency is crossed, the data will be downsampled to a rate of 1 minute for 21 days and so on.

Grafana

Grafana is an HTML5 data visualisation dashboard that has been developed as a fork of the original Kibana code base. Originally designed for Graphite. The visualisations that Grafana produces are very effective, and most allow the user to interactively explore the data in Graphite.[gra, a]

Development and Implementation

As a part of the project, various changes to the monitoring architecture were made with the aim of reducing or eliminating most of the shortcomings of the existing architecture. In the following section of this document, the changes will be explored and summarized.

Addition of new output to MAE

As mentioned previously, the Cockpit dashboard used in CASTOR has high latency and does not support a structured metric format. To ensure the elimination of this problem, Graphite service came into picture. The cockpit is built in Django framework and to implement the output to Graphite in addition to Cockpit, thorough study of the CASTOR code base has been done.

Once Graphite had been deployed on a test instance and verified to work correctly, it was pertinent to consider which protocol to use for sending the metric data to it. A small amount of research led to the fact that the plaintext protocol would be suitable.[gra, b]

To be able to parse a log file effectively, Graphite must know the format of each log message - the position of each field, the datatype of each field, and so on. The process of developing a feeder to parse log files involves writing a significant number of regular expressions to be able to parse each entry - a challenge made more difficult by the inconsistent and complex nature of the logging infrastructure within the Cockpit and MAE.The metric format followed by Graphite is as follows [mon]:

```
metricPath value timestamp
```

The metric data had to be parsed properly to obtain the desired format. An output plugin has been developed and tested on c2mon02 machine in production. During the course of several weeks, it proved to be able to cope well with the constant heavy flow of data.

Migration of historic data

After Graphite had been tested, the historic data had to be collected in the Graphite database (whisper). The old MySQL database for Cockpit is present in the form of Django models and contains around 88 million data points over a span of 15 years. There were several files that had to be collected, with most files containing messages with different log formats and unwanted special characters due to different information in the messages. This posed a difficult problem when it came to extracting useful information from each file. All this data resided on c2mon02 and the processing has been done using custom admin django commands to integrate the stand-alone script into the existing CASTOR django framework. The metrics which were chronologically sorted earlier, had to be extracted into JSON format and parsed using regular expressions.[dja]

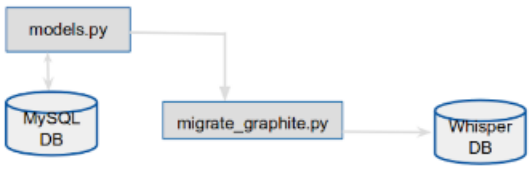

Figure 3: Migration schema

```
#prepare for graphite format : path value timestamp
def _flatten_dict(self, input, ts, sep='.', prefix=None):
    for name, value in sorted(input.items()):
    #concatenate prefix
    fullname = sep.join(filter(None, [prefix, name]))
    #check for nested dictionaries
    if isinstance(value, dict):
        for rslt in self._flatten_dict(value,ts,sep,fullname):
            yield rslt
    else:
    #ignore metrics with special characters
        if re.match(r"^[\w\.\-\+\=]+$", fullname):
            try:
                yield '%s%f%d\n' % (fullname, value[0], ts)
            except TypeError:
                pass
            else:
                self.stderr.write("IgnoringMetric: '%s' (invalid)\n" %
fullname)
```

Listing 1: Metric data parsing

Metric collection for MAE-Spark

As the MAE-Spark replaced the old MAE for computing the metrics, a lot of useless metrics were dropped. A new output plugin has been developed thereafter. It was pretty easy to write the new plugin because the old code for the old MAE could be reused. The new plugin has been added to the spark job wherein it runs as a part of it. The spark job has been tested on only one data producing instance which is not capable of producing data for all metrics. Consequently, only some metrics have been tested for data ingestion in Graphite database.

Visualisation

1. Graphite Web

   The graphite dashboard interface displays more than one graph at a time, with all graphs showing the same time range. Unless you're using the HTTP interface to embed graphs in your own applications or web pages, this is the Graphite interface you'll use most often. However, the user interface of Graphite isn't particularly user-friendly and the charts lack interactivity, which is something that many other HTML5 visualisation dashboards can provide. [gra, c]

2. Grafana

   It contains a unique Graphite target parser that enables easy metric and function editing. A test instance created log data for which sample dashboard were created. It rendered fast client side rendering (even over large time ranges) using Flot with a multitude of display options (Multiple Y-axis, Bars, Lines, Points, smart Y-axis formats and much more). [gra, a]

3. Performance

   Regarding performance, both Grafana and Graphite interface perform really well when generating comparatively large plots. Grafana provides interactive graphs and still performs reasonably well for larger datasets. The difference in the performance of Cockpit and Grafana is significant. The Cockpit dashboard gives timeout error if triggered to load and plot a 3 year old data point. On the other hand, Grafana can load a similar point in 1.09 seconds.

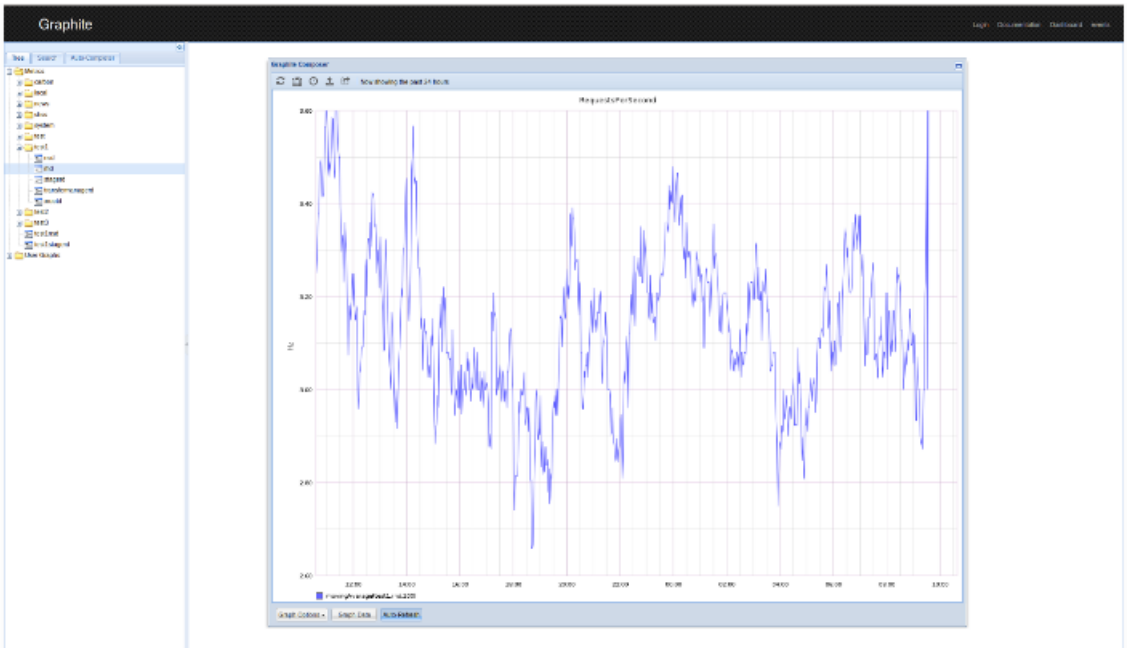

Figure 4: Graphite web dashboard

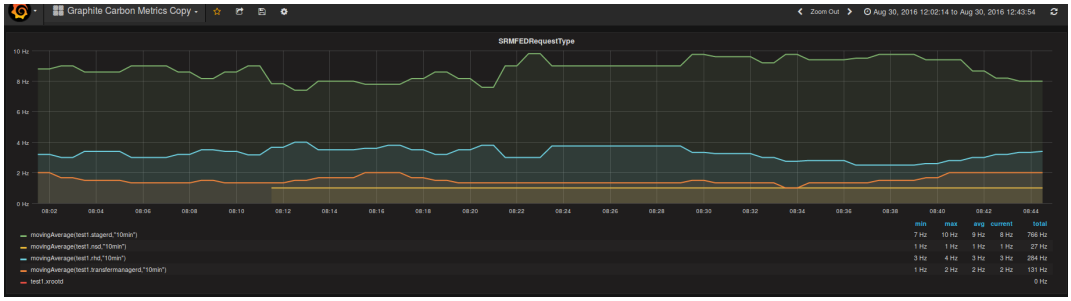

Figure 5: Grafana dashboard

Reproducibility

The output plugin for the old MAE has been packaged as an RPM and in CERN's repositories - was slowly accepted into CERN's CASTOR production environment. After being tested during the course of several weeks on a representative machine c2mon02, it has proven to be stable, lightweight and resilient – performing well and coping gracefully with service downtime. The output plugin for the MAE-Spark is to be packaged along with MAE-Spark to run as a spark job on the analytix cluster of hadoop services of CERN.

Architecture

This section shows the revised infrastructure of CASTOR mainly on the output side.

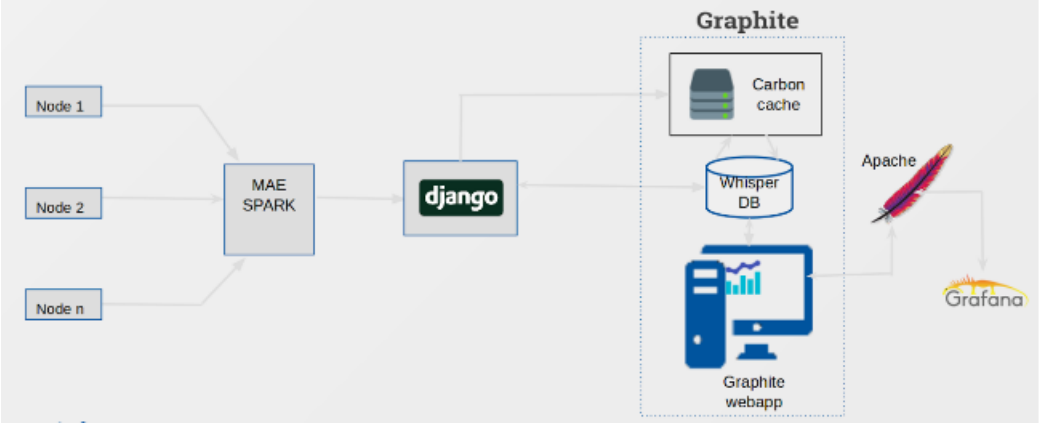

Figure 6: New Infrastructure

Project summary

Conclusion

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.The verbose codebase of cockpit has been replaced by the flexible Graphite scripts. To conclude, the project has been largely successful. The existing logging infrastructure had several improvements made to it - GraphitePusher is now used in place of cockpit pusher meaning that log monitoring is more resource-efficient, and therefore can also run on less powerful machines. However, the Graphite service has to be managed by the group until CERN IT Service comes up with the time-series database service in the future. One of the problems is that the spark streaming into Graphite leads to memory overflow. 95% time is spent in garbage collection. This can be improved by passing the metrics.properties file in the spark submit command.

Future Work

Regarding the future plan for the work undertaken in this project, the testing of the proposed Graphite infrastructure will continue. An internal review has showed positive feedback for the new infrastructure, and the Grafana dashboards were also praised as being useful. Regarding the future of Graphite, an internal review showed that it has been indeed useful, and that it may provide a great deal of value over the evolving InfluxDB system in the department, and so evaluation will continue.

References

Metrics analytic engine. URL https://twiki.cern.ch/twiki/bin/viewauth/DSSGroup/MetricsAnalysisEngine.

Castor. URL http://castor.web.cern.ch/.

Cockpit. URL https://twiki.cern.ch/twiki/bin/viewauth/DSSGroup/CockpitWebInterface.

Django. URL https://docs.djangoproject.com/en/1.10/howto/custom-management-commands/.

Grafana, a. URL http://grafana.org/.

Graphite, b. URL https://github.com/tmm1/graphite.

Graphite docs, c. URL http://graphite.readthedocs.io/en/latest/index.html.

Monitoring with graphite. URL https://www.safaribooksonline.com/library/view/monitoring-with-graphite/9781491916421/ch01.html.

Appendix

Detailed Previous Infrastructure