Monitoring performance of a highly distributed and complex computing infrastructure in LHCb

Z Mathe¹, C Haen¹ and F Stagni¹
¹ CERN, European Organization for Nuclear Research, Switzerland
E-mail: zoltan.mathe@cern.ch

Abstract. In order to ensure an optimal performance of the LHCb Distributed Computing, based on LHCbDIRAC, it is necessary to be able to inspect the behavior over time of many components: firstly the agents and services on which the infrastructure is built, but also all the computing tasks and data transfers that are managed by this infrastructure. This consists of recording and then analyzing time series of a large number of observables, for which the usage of SQL relational databases is far from optimal. Therefore within DIRAC we have been studying novel possibilities based on NoSQL databases (ElasticSearch, OpenTSDB and InfluxDB) as a result of this study we developed a new monitoring system based on ElasticSearch. It has been deployed on the LHCb Distributed Computing infrastructure for which it collects data from all the components (agents, services, jobs) and allows creating reports through Kibana and a web user interface, which is based on the DIRAC web framework. In this paper we describe this new implementation of the DIRAC monitoring system. We give details on the ElasticSearch implementation within the DIRAC general framework, as well as an overview of the advantages of the pipeline aggregation used for creating a dynamic bucketing of the time series. We present the advantages of using the ElasticSearch DSL high-level library for creating and running queries. Finally we shall present the performances of that system.

1. Introduction

DIRAC interware [1, 2] is a software framework, which delivers a complete solution for managing distributed heterogeneous computing resource, such as Grid, Cloud and allows users the efficient use of these resources. LHCbDIRAC is the LHCb extension of DIRAC, which implements the LHCb specific computing requirements. Within LHCb more than 134 service and 146 agent instances are running, which requires real time monitoring in order to discover problems and take different actions for example: restart a service, stop a certain production or notify a certain Grid site in case of failure, remove a certain storage element from production or notify the appropriate person. In addition monitoring of different activities such as data analysis, data processing, data replication, data merging, data removal and Monte Carlo simulation, is also crucial in order to be able to monitor each activity and to keep track of the usage of the computing resources.

The DIRAC Activity monitor is based on round-robin database tool (RRD) [3] and the Accounting System [4], are developed to fulfil these criteria. The current Accounting is implemented in MySQL and is not optimised for analysis of time series data. The Activity monitor can not be easily extended and in addition it is not very simple to perform analysis on the RRD files.
We studied the possible replacement of the current systems in a previous paper [5]. The result of the study is the new Monitoring system for real time monitoring and analysis, which collects data from all the components: agents, services and DIRAC tasks; and allows creating reports through Kibana and the Accounting web application, which is based on the DIRAC web framework. Accounting web application is the official user interface, because it:

- is user friendly; it provides very simple selectors
- is integrated to DIRAC; it does not require to learn an extra tool
- provides all functionalities what Kibana has

In addition to these advantages it allows to create high quality plots.

In this paper we present the new Monitoring system, which is based on Elasticsearch distributed search and analytic engine and designed to store time series data. In section 2 we present the monitoring system architecture and we give a detailed description of the back end and front end as well as a short description of technologies used to develop the system. The way monitoring data is stored presented in section 3. The dynamic bucketing, which is used during the plot creation is described in subsection 3.1. Finally, the performance of the new monitoring system is presented in section 4.

2. The Monitoring system architecture
The architecture of the system is shown in Figure 1. The core of the system is the back end, which consists of different components used to store and retrieve the data. The user can interact with the system via the front end.

2.1. Back end
It is implemented in python using different libraries: Elasticsearch-dsl [6] for interacting with the Elasticsearch, Matplotlib [7] plotting library for creating plots, RabbitMQ [8] message broker software for fail over and other functionalities which are provided by DIRAC. We are using an ElasticsearchDB module which is a wrapper around the Elasticsearch-dsl, which exposes the database functionalities through a developer friendly interface. This module is used by the Monitoring DB util which implements specific queries used to retrieve the data.

The Monitoring type is a python configurable object, which describes the monitoring data content. The Monitoring type contains the following elements:

- Data retention: The system keeps the data, which is not older than a certain period. For example: last two months. Currently, the data which is one year older will be deleted by the Monitoring service using a dedicated thread. This thread is responsible for monitoring each monitoring type and take action taking into account the data retention.
- Key fields: these are selectable conditions
- Monitoring fields: these are values which will be plotted.

Each Monitoring types has an associated Plotter, which provides information such as type of the plot, title of the plot, x-axis, y-axis, plot colors and data to be plotted during the plot creation. The MainReporter is used to create plots using the appropriate Plotter and the Monitoring DB util. It is based on the DIRAC Graph package which is a wrapper around the Matplotlib. Currently, the system provides two different plotters: ComponentMonitoring and WMSMonitoring. Both components are used to create stacked line, pie chart and vertical bar plots. Plots are created using two level caching mechanisms in order to improve the performance of the system:

- DataCache: data used to create the plots are kept in memory. Consequently, we can create different plots using the same data without querying the database.
Figure 1. Architecture of the system

- FileSystem: the plots which are created by the Plotter are kept in the File System. Consequently, the plots which are already created, will be returned to the user without interacting to the database.

The Monitoring service is the highest component of the back end. It exposes the low level functionalities through the DIRAC Secure Transport (DSET) protocol [9, 10] to the front end.

2.2. Front end

It consists of different components: Accounting WebApp, Monitoring client and MonitoringReporter.

The MonitoringReporter inserts time series data into the database. It is used by different components of DIRAC:

- StatesMonitoringAgent: This agent retrieve the information from the DIRAC Workload Management system, transform the data to the appropriate data format and insert it to the database.
- SystemAdministrator service: This service keeps track and allows remote management of the installed services. It retrieve the information such as CPU, Memory, number of running threads for a given service and insert it to the database.

It provides two level fail over mechanism using memory and RabbitMQ in order not to lose the data in case of problem.
The Monitoring client exposes the Monitoring service functionalities to the high level components, which are used by the users. The Monitoring client is used by the Monitoring web handler which are developed within the DIRAC web framework and it is used to serve the user requests coming from the web browser. The DIRAC web framework [11] provides all functionalities used to visualize the plots:

- Authentication and authorization
- Customization
- Plot sharing mechanism

The Accounting WebApp shown in figure 2 is based on the DIRAC web framework and is the main user interface for Accounting and Monitoring. The front end of the Monitoring system is integrated to the already existing Accounting WebApp. In addition to the Accounting WebApp, we have setup Kibana [12] for experts.

3. Data storage

The data format is key/value pairs defined by the Monitoring type. Each monitoring activity is recorded in JSON (JavaScript Object Notation) format. The records have the following structure:

\[
\{\text{timestamp} : t, \text{key}_1 : \text{value}_1, \text{key}_2 : \text{value}_2, \ldots \text{key}_n : \text{value}_n, \text{mkey}_1 : \text{mvalue}_1, \ldots \text{mkey}_n : \text{mvalue}_n\}
\]

where \(\text{key}_1 : \text{value}_1, \ldots \text{key}_n : \text{value}_n\) are the selectable conditions, \(\text{mkey}_1 : \text{mvalue}_1, \ldots \text{mkey}_n : \text{mvalue}_n\) are the monitoring fields which values will be plotted at a given \(t\) time.

The monitoring records are stored in daily indexes and have a very well defined format: "systemsetup-monitoringtype-index-year-month-day". For example: lhcb-wmsmonitoring-2017-01-18. The data is stored by an Elasticsearch cluster provided by CERN centralised Elasticsearch service.
3.1. Data retrieval

Metric bucket and pipeline aggregations are used to retrieve the data points. Sometimes it is not enough to perform a single aggregation like average, sum on the raw data and it requires to execute some pre computation instead of using a single aggregation. For example the table 1 contains the information of different tasks. The table may contains many duplications, because it can happen that the same data is registered more than once. Performing a single aggregation such as average number of running tasks on the data which is in table 1 where the bucket length is 18 minutes is returning wrong values:

- $\text{bucket}_1 \text{SUM}(09:00) + \text{SUM}(09:15) = 5$
- $\text{bucket}_2 \text{SUM}(16:15) = 7$
- $\text{bucket}_3 \text{SUM}(16:30) = 1$

In order to calculate the correct values we have to perform an extra aggregation on the data:

- $\text{bucket}_1 \text{AVG}($SUM(09:00) + SUM(09:15)) = 2.5$
- $\text{bucket}_2 \text{AVG}($SUM(16:15)) = 7
- $\text{bucket}_3 \text{AVG}($SUM(16:30)) = 1

We take advantages of the pipeline aggregation in order to create dynamic buckets. Figure 3 shows an overview of the Dynamic bucketing. We call dynamic buckets the result of Aggregation1, because they are created taking account various conditions and only kept in memory for a short period. The output of the Aggregation1 is the input of the Aggregation2. The result of the Aggregation2 aggregation will be used to create the plots. Because of the Dynamic bucketing we do need to perform any streaming or pre computation on the raw data.
4. Performance of the system

- Running jobs grouped by sites
- Running jobs grouped by job group
- Running jobs grouped by job type
- Waiting jobs grouped by activities
- Memory usage by host
- Memory usage by component

Figure 4. Six different plots used to test the system; The a, b, c, d plots are for monitoring the WMS, while e, f plots are component monitoring plots.

In our tests we used an Elasticsearch 2.3.3, which has 8 nodes provided by CERN centralised Elasticsearch service. The Elasticsearch cluster [13] is using 8 virtual machines (VMs) provided by the CERN OpenStack [14]:
Figure 5. Performance of the system

- 3 master nodes with 2 processors with 4 GB of Memory, running on CentOS 7
- 2 search nodes with 4 processors with 8 GB of Memory, running on CentOS 7
- 3 data nodes with 32 processors with 60 GB of Memory, running on CentOS 7

The hypervisors of the VMs, which are dedicated to the master and search nodes are configured with SSD disk only. The data nodes are running on hypervisors, where the hyper threading is enabled. We used about 500 million records which have been recorded in 2.5 month. We defined six different queries shown in figure 4. In order to ensure that the plots will be not cached we randomly generated the time intervals of the queries between 1 and 75 days.

We generated high workload using 20 "virtual users" (python threads), which had been simultaneously running in parallel during 7200 seconds. We developed a simple python script, which is executed by the python threads. Using the Monitoring client, this simple python script had been simulating the users behaviour by generating and executing the database queries. In our tests we measured the response time and the throughput which are shown in Figure 5 in the client level (Monitoring client), which does not measure the display time of the plot. The average response time of the system is 33.37 second shown in Figure 5(a). We measured the throughput shown in Figure 5(b), which is the number of transactions during 7200 seconds.

Taking into account that the users are not behaving the same way as our performance tests, because the are not generating random plots for a very long time intervals in our case 75 days, we can conclude that the system performed very well.

5. Conclusion
Elasticsearch is fulfilling our needs for handling huge amount of time series data and can be used for any type of monitoring purpose. Kibana is a very good tool to create complex dashboards, but it is not possible to create very good quality plots and it requires experience. Consequently, we developed an extension of the Accounting web application which allows users to create plots using the Monitoring system. We measured the performance of the system, which shows that the system fulfil our requirements. The new Monitoring system is fully operational in production and we are evaluating the other use cases in order to improve the DIRAC performance and the possible improvements of the DIRAC component monitoring.

References
[1] Casajus A, Ciba K, Fernandez V, Graciani R, Hamar V, Mendez V, Poss S, Sapunov M, Stagni F, Tsaregorodtsev A and Ubeda M 2014 Journal of Physics: Conference Series 396 032107 URL http://stacks.iop.org/1742-6596/396/i=3/a=032107
[2] Tsaregorodtsev A 2014 *Journal of Physics: Conference Series* 513 032096 URL http://stacks.iop.org/1742-6596/513/i=3/a=032096

[3] Rrd URL www.rrdtool.org/

[4] Casajus A, Diaz R G, Puig A and Vazquez R 2011 *Journal of Physics: Conference Series* 331 072059 URL http://iopscience.iop.org/1742-6596/331/7/072059/pdf/1742-6596_331_7_072059.pdf

[5] Mathe Z, Casajus A, Stagni F and Tomassetti L 2015 *Journal of Physics: Conference Series* 664 042036 URL http://iopscience.iop.org/article/10.1088/1742-6596/664/4/042036/pdf

[6] Elasticsearch dsl URL http://elasticsearch-dsl.readthedocs.io/en/latest/index.html

[7] Matplotlib URL http://matplotlib.org

[8] Rabbitmq URL https://www.rabbitmq.com

[9] Diaz R G and Casajus A 2007 *Journal of Physics: Conference Series* URL https://indico.cern.ch/event/3580/contributions/1768562/attachments/712678/978396/CHEP2007_-_Adrian_Casajus.pdf

[10] Diaz R G and Casajus A 2010 *Journal of Physics: Conference Series* URL http://iopscience.iop.org/article/10.1088/1742-6596/219/4/042033/pdf

[11] Mathe Z, Casajus A, Stagni F and Lazovsky N 2015 *Journal of Physics: Conference Series* 664 062039 URL http://iopscience.iop.org/article/10.1088/1742-6596/664/6/062039/pdf

[12] Kibana URL https://www.elastic.co/products/kibana

[13] private communication

[14] Bell T, Bompastor B, Bukowiec1 S, Leon J C, Denis1 M K, van Eldik J, Lobo M F, Alvarez1 L F, Rodriguez D F, Marino A, Moreira B, Noel B, Oulevey T, Takase W, Wiebalck A and Zilli S 2015 *Journal of Physics: Conference Series* 664 022003 URL http://iopscience.iop.org/article/10.1088/1742-6596/664/2/022003/pdf