ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

MONITORING THE NEW ALICE ONLINE-OFFLINE COMPUTING SYSTEM

A. Wegrzynek*, V. Barroso, CERN, Geneva, Switzerland
G. Vino, INFN, Bari, Italy
for the ALICE Collaboration

Abstract

ALICE (A Large Ion Collider Experiment) is a particle detector designed to study heavy-ion collisions and the physics of strongly interacting matter and the quark–gluon plasma at the CERN LHC (Large Hadron Collider).

ALICE has been successfully collecting physics data since 2010. Currently, it is in the preparations for a major upgrade of the computing system, called O^2 (Online-Offline) and scheduled to be deployed during Long Shutdown 2 in 2019–2020.

The O^2 system will consist of 268 FLPs (First Level Processors) equipped with readout cards and 1500 EPNs (Event Processing Node) performing data aggregation, calibration, reconstruction and event building. The system will readout 27 Tb/s of raw data and record tens of PBs of reconstructed data per year.

To allow an efficient operation of the upgraded experiment, a new Monitoring subsystem will provide a complete overview of the O^2 computing system status, detect performance degradation or component failures. The ALICE O^2 Monitoring subsystem will collect and receive up to 600 kHz of metrics. It will consist of a custom monitoring library and a toolset to cover four main functional tasks: metric collection, metric processing, storage, visualization and alarming.

This paper describes the Monitoring subsystem architecture and the feature set of the monitoring library. It also shows the results of multiple benchmarks, essential to ensure that the processing and storage performance requirements are met. In addition, it presents the evaluation of preselected tools for each of the functional tasks, including Collectd, Apache Flume, Apache Spark, InfluxDB and Grafana. It concludes by describing the next steps towards the final subsystem.

INTRODUCTION

The ALICE Experiment

ALICE (A Large Ion Collider Experiment) [1] is a heavy-ion detector designed to study the physics of strongly interacting matter (the Quark–Gluon Plasma) at the CERN Large Hadron Collider (LHC). ALICE consists of a central barrel and a forward muon spectrometer, allowing for a comprehensive study of hadrons, electrons, muons and photons produced in the collisions of heavy ions. The ALICE collaboration also has an ambitious physics program for proton–proton and proton–ion collisions.

After a successful Run 1 ALICE has been taking data in Run 2 since the beginning of 2015. In the end of 2018 the LHC will enter into a consolidation phase – Long Shutdown 2. At that time ALICE will start its upgrade to fully exploit the increase in luminosity.

The upgrade foresees a complete replacement of the current computing systems (Data Acquisition, High-Level Trigger and Offline) by a single, common O^2 (Online-Offline) system.

The ALICE O^2 system

The ALICE O^2 computing system [2] will allow the recording of Pb–Pb collisions at 50 kHz interaction rate. Some detectors will be read out continuously, without physics triggers. Instead of rejecting events the O^2 system will compress the data by online calibration and partial reconstruction.

The first part of this process will be done in dedicated FPGA cards that receive the raw data from the detectors. The cards will perform baseline correction, zero suppression, cluster finding and inject the data into the memory of the FLP (First Level Processors) to create a sub-timeframe. Then, the data will be distributed over EPNs (Event Processing Node) for aggregation and additional compression.

The O^2 facility will consist of 268 FLPs and 1500 EPNs. Each FLP will be logically connected to each EPN through high throughput links. The O^2 farm will receive data from the detectors at 27 Tb/s, which after processing will be reduced to 720 Gb/s.

OBJECTIVES DEFINITION

The Monitoring subsystem is part of O^2 and provides comprehensive functionality in metric collection, processing, storage, visualization and alarming as shown in Fig. 1. Three already short-listed solutions are being evaluated: MonALISA [3], Modular Stack (see MODULAR STACK section) and Zabbix [4]. This paper aims to provide details and performance measurements of the Modular Stack.

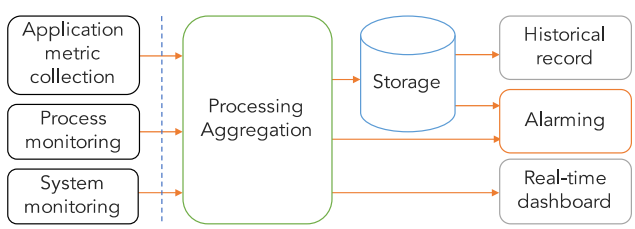

Figure 1: Functional architecture of the Monitoring subsystem.

* E-mail: adam.wegrzynek@cern.ch

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

Overview

The O² Monitoring subsystem collects three classes of metrics:

- Application.
- Process.
- System (and infrastructure).

Client side metrics are pushed to the processing and aggregation backend, and then written into permanent storage. Some selected metrics are published for alarming and real-time visualization. The stored metrics can be browsed and plotted in the historical record dashboard.

System and Infrastructure Monitoring

The System monitoring provides probes to various operating system metrics regarding for example:

- CPU.
- Memory.
- Network.
- Storage.
- Hardware status.

It can also query devices such as network switches, routers and power supplies via standardized or generally available protocols to obtain their current status.

System monitoring should be compatible with the CERN CentOS 7 and support other UNIX based systems on a best effort basis.

Process Monitoring

The Process monitoring collects performance metrics of each O^2 process such as:

- CPU usage.
- Memory usage.
- Bytes sent and received per network interface.
- Context switches count.
- Open file descriptors count.

It can be implemented either as a library linked to the process or as an external daemon running on each host.

Application and Metric Collection

The Application metric collection provides an entry point from O^2 processes to the Monitoring subsystem. It forwards user defined metrics to the processing backend via connection or connection-less transport protocols.

Metric Aggregation and Processing

The Metric aggregation and processing correlates and manipulates metrics coming from different origins. The processing may occur at any step of the monitoring chain, including the central collector if correlations between widely different metrics are needed.

The processing task types are the following:

- Data suppression (e.g. for link status, only store transitions on/off and off/on).
- Data enrichment (e.g. add tags).
- Data aggregation (e.g. cumulative metric for all FLPs of a given detector).
- Data correlation (e.g. detect abnormal situations).

The output metrics are filtered and routed to storage, real-time dashboard and alarming.

Storage

The Storage receives and writes metrics into an historical record. It must support large input metric rates. It accepts queries to retrieve stored metrics. It also provides administration tools to manage its internal parameters.

Given that the O^2 Monitoring subsystem will receive gigabytes of metrics daily, storage needs to support archiving and downsampling – aggregating metrics in time to reduce their overall size.

Visualization

The visualization dashboards display metrics in form of plots, gauges, bars and data tables. They can provide views for different purposes:

- Near-real-time – for shift crews, providing a summary view of the ongoing ALICE operations; low latency is of extreme importance.
- Historical record – for experts, allowing for drill down and detailed views.

Dashboards can easily be accessed on various operating systems and outside of the ALICE Point 2.

Alarming

The Alarming scans metrics passing through the monitoring system and detects abnormal situations: thresholds exceeded, value not present or more advanced detector and/or experiment specific logic.

Two different types of alarming implementations are possible:

- Late stage alarming – based on historical records by querying the storage.
- Online alarming – scanning metrics directly during processing.

REQUIREMENTS

The list of requirements regarding the monitoring subsystem has been established from the information available in the O^2 Technical Design Report [2]. Each solution must meet the following mandatory requirements:

- Compatible with the O² reference operating system (currently CERN CentOS 7).
- Well documented.
- Actively maintained and supported by developers.
- Run in isolation when external services and/or connection to outside of ALICE are not available.
- Capable of handling 600 kHz input metric rate.
- Scalable to >> 600 kHz if necessary.
- Handle at least 100 000 sources.
- Introduce latency no higher than 500 ms up to the processing layer, and 1000 ms to the visualization layer.
- Impose low storage size per measurement.
- Aligned with functional architecture specified in OBJECTIVE DEFINITION section:
  - System sensors.
  - Metric processing.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

- Historical record and near-real-time visualisation.
- Alarming.
- Storage that supports downsampling.

In addition, some optional requirements may positively influence the final rating:

- Supported by CERN or used in one of the experiments/departments.
- Self-recovery in case of connectivity issues.

MODULAR STACK

The Modular Stack solution aims at fulfilling the requirements specified in the REQUIREMENTS section by using a set of open source tools. Such approach enables the possibility of replacing one or more of the selected components in case alternative options provide improved performance or additional functionalities.

The selected component responsible for retrieving system metrics (related to CPU, memory and I/O) is Collectd [5]. These metrics together with the application defined metrics require a high-performance collection and multiplexing.

A tool that can cope with such task is Apache Flume [6], a distributed service that moves large amount of monitoring data from the O^2 processes in an efficient way. Flume supports numerous data formats and also provides an API to develop custom components [7] for extra functionality:

- Source – parses received data into Flume events.
- Sink – parses Flume events into any implemented format.
- Interceptor – component attached to a Source that can modify Flume events.

The connectionless UDP protocol was selected to receive metrics. Contrary to TCP, UDP has no operating system limitation regarding the number of sources.

Flume accomplishes simple processing tasks (e.g. data suppression and data enrichment) while the more complex computing is executed by Apache Spark [8], "a fast and general-purpose engine for large-scale data processing".

As a next step the metrics are pushed to an InfluxDB database [9] which is optimized to store time series data points. It also provides high performing writing, "expressive SQL-like query language tailored to easily query-aggregated data" and low disk occupancy per measurement – three bytes for non-string values. The InfluxDB engine supports downsampling via Retention Policy and Continuous Queries. The combination of these two features requires only the time resolution and time period to be specified (e.g. store 1 data point per 30 seconds for data not older than 30 days).

Grafana [10] has been chosen as data visualisation tool. It supports both real-time and historical record dashboards. It can also generate alarms based on values coming from the database.

Riemann [11] is used as the main alarming tool. It inspects metrics on the fly and generates notifications when undesired behaviour is detected.

The selected tools work without the need for external services or internet connectivity. They provide extensive user and developer documentation. They are also compatible with most of the UNIX based operating systems, including CERN CentOS 7.

All Modular Stack tools are being used in production by the CERN IT team to monitor data centres and develop experiment dashboards [12]. This ensures that an increasing number of teams at CERN will gain the experience in working with these tools and provide potentially valuable feedback for the final O^2 Monitoring.

MONITORING LIBRARY

The O^2 Monitoring library [13] covers two tasks: process monitoring and application metric collection. The library can transport values as integers, floating point numbers, strings and long integers. It supports multiple server side backends:

- MonALISA (UDP via ApMon library).
- InfluxDB (UDP and HTTP).
- Flume (UDP and HTTP).
- O^2 Logging (custom protocol).
- Zabbix (Zabbix protocol).

The library allows gathering process related metrics such as: uptime, CPU and memory utilization, bytes sent and received per interface. It features calculations of derived values such as rate and average. It also allows appending a metric with metadata (tags) and send multiple values in a single transaction.

RISK ASSESSMENT

The Modular Stack requires maintaining multiple tools and therefore compatibility between them. This results in higher system complexity and necessity to acquire knowledge on all the components. In case one of the selected tools breaks backward compatibility, becomes obsolete or its maintenance or support is dropped, the system might need to be adjusted or even redesigned. On the other hand, only standardized protocols are used for the communication which can facilitate any future migration.

There is also the possibility that newly introduced features will require the purchasing of a subscription or license.

PERFORMANCE TESTS

Test Specification

One of the mandatory requirements is the capability of handling 600 kHz of metric points coming from 100 000 sources. This requirement should be verified under the following test scenarios:

- No processing – sending data directly to the storage.
- With processing:
  - Pass-through – forward metric to the storage.
  - Edit a metric – modify one of field of the metric.
  - Aggregation – aggregate metrics of the same origin and type coming in a predefined time period and calculate the average (simulate aggregation of values coming from different detectors).

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

For each scenario, the following parameters should be measured:

- Maximum metric and transaction rate that processing and storage can cope with; it can also be presented as % of successfully processed/stored metrics as a function of overall input metric rate.
- Latency between collecting (benchmark application timestamp) and displaying a metric (when we can see it).
- Latency between collecting and processing a metric (when we can act on it).
- Identify what is the limiting factor (e.g. CPU, memory, network etc.).

The tests should be launched in the reference set-up that consists of three machines equipped with Intel E5-2640 v3, 40 GbE and SSD drives.

Test procedure

The test procedure is semi-automatized and allowing for a test to be quickly repeated in any of the configurations. The benchmark is based on the O^2 Monitoring library and can be deployed and controlled via Ansible [14]. Flume publishes counters of sources and sinks which are available as JSON formatted strings via the internal HTTP server. Custom made scripts read out the Flume counters, probe InfluxDB to reveal the number of successfully stored metrics and write these values in the dedicated database. Eventually all statistics are displayed with Grafana.

The latency was measured by passing a metric through the system and inserting a timestamp at each step. A script transforms these values into histograms that can be easily viewed in Grafana. To handle the clock synchronization issue, the benchmark and the storage were started on the same machine.

Performance notice

The Linux kernel uses interrupts to process UDP packets coming from NIC (Network Interface Controller). When dealing with high packet rate the application running on the same core as these interrupts suffers a decrease in performance because of the large number of context switches.

The Linux scheduler (CentOS 7, 3.10.0-514.26.2.el7.x86_64) tends to move processes and threads between CPU cores to optimize their usage but does not take into account the influence of network interrupts. During the tests, the benchmark application was moved between CPUs. As some of them were handling interrupts, periodical performance drops were observed.

Summarising, it is crucial to choose separate CPU cores for the network interrupts and the application itself but keeping in mind to stay on the same NUMA (Non-uniform memory access) node to avoid inter-CPU bus penalty.

Metric rate

The Modular Stack was tested in four different configurations to fully understand the impact of each additional component.

Configuration 1 – Benchmark to Flume

Configuration 1, as shown in Fig. 2, includes benchmark application sending metrics to a Flume custom UDP source and then having a null sink dropping them.

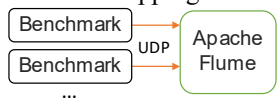

Figure 2: Benchmark to Flume configuration.

In this simplified configuration the impact of the NUMA, the network card interrupts and UDP socket count was measured. In addition, a test with a custom timestamp interceptor that extends each metric with the current timestamp was made.

Table 1: Metric Rate with Different NUMA and Flume Configurations

Number of UDP sources	Flume interceptor	NUMA node	Metric rate [kHz]
1	-	0 (interrupts)	46
	-	0 (no interrupts)	121
	-	1	112
1	timestamp	0 (no interrupts)	106

As mentioned in the *Performance notice* subsection, Flume running on the same core as NIC interrupts could handle rates two and a half times lower than running them on a separate CPU, what can be observed in the first two rows of Table 1.

The introduction of the Flume timestamp interceptor degrades metric rate by roughly 10%.

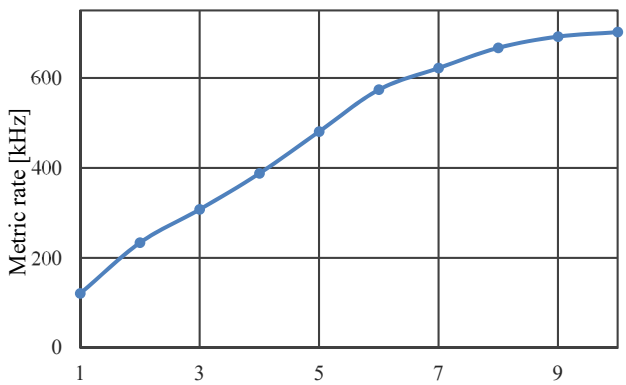

Figure 3: Metric rate as a function of the Flume UDP sources.

As each Flume component (e.g. source or sink) runs in a single thread, it is limited by the performance of a CPU core. Therefore, increasing the number of components provides an almost linear performance increase until the NIC receiving limitation is hit – see Figure 3. The required 600 kHz rate was reached using a single instance of Flume with 7 UDP sources.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

In addition, Flume (as well as InfluxDB) can handle multiple metrics per single UDP packet (or so-called measurement). The Figure 4 shows metric and transaction rate as a function of the metrics per measurement for 1 UDP source. This parameter also scales linearly, therefore the final system may require less UDP sources than initially estimated.

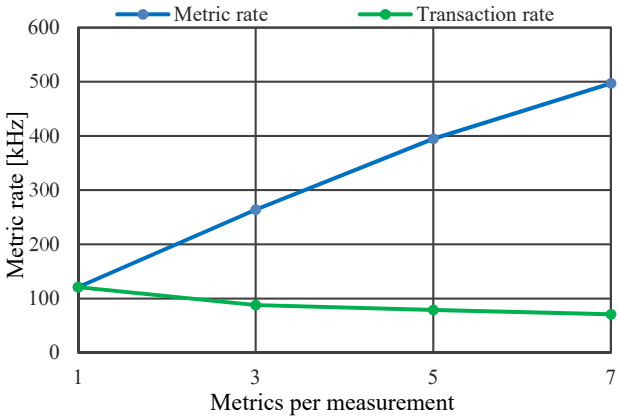

Figure 4: Metric and transaction rate as function of number of metrics per measurement.

Configuration 2 - Benchmark to InfluxDB

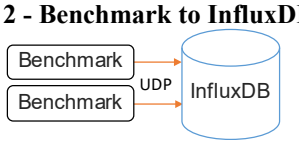

Figure 5: Benchmark to InfluxDB configuration.

The configuration shown in Fig. 5 consists of benchmark application sending metrics directly to the database and has been used to estimate the writing capability of the InfluxDB engine. Figure 6 presents the percentage of successfully stored metrics as a function of the metric rate in three different configurations:

- 1 UDP listener / HDD drive.
- 3 UDP listeners / HDD drive.
- 3 UDP listeners / SSD drive.

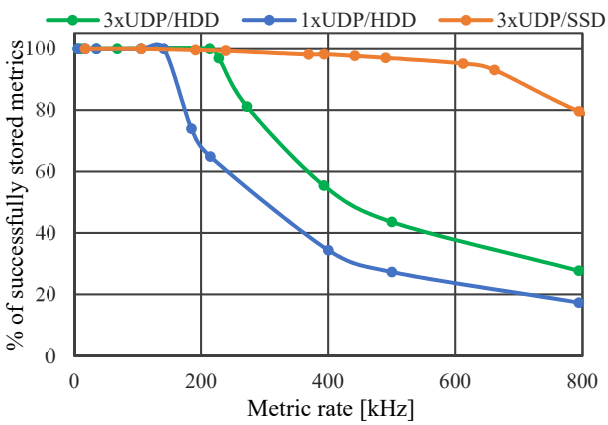

Figure 6: Percentage of stored metrics as a function of metric rate.

The HDD setup reached 142 kHz with a single UDP listener and 216 kHz with 3 UDP listeners. Further increase in the number of listeners did not have any major impact on the results. As expected, a sharp drop is observed for higher rates.

The SSD setup behaves differently. The percentage of stored metrics decreases slowly, which is not yet fully understood. At 300 kHz rate 1% of the metrics are lost and 5% at 480 kHz.

Configuration 3 – Benchmark-Flume-InfluxDB

The chain Benchmark-Flume-InfluxDB (see Fig. 7) was tested using multiple metric streams and SSD disks only. Each metric stream consists of a dedicated Flume source, sink and InfluxDB listener. All listeners ran on the same database instance. The impact of the interceptor was also taken into account. Four streams provided 400 kHz rate with acceptable metric loss – the results are presented in Fig. 8. Increasing the number of streams resulted in significant metric drop. The measured value is lower than the specified 600 kHz, although it should be sufficient as not all the metrics need to be stored in the database (e.g. many metrics will be dropped due to data suppression) .

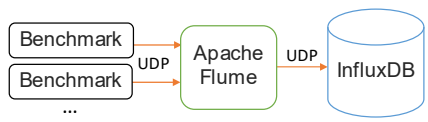

Figure 7: Benchmark through Flume to InfluxDB configuration.

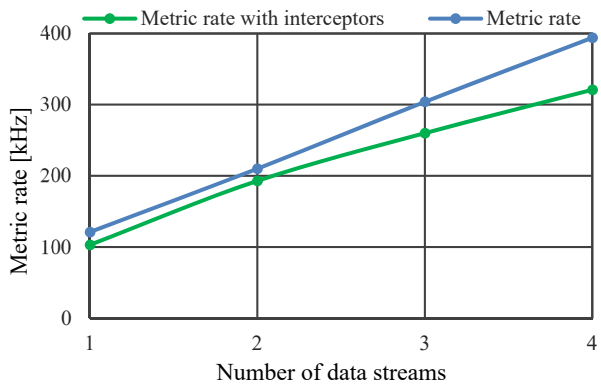

Figure 8: Metric rate as a function of data streams

Configuration 4 – final configuration (no alarming)

The final configuration accommodates, in addition to Configuration 3, Apache Spark applying the batch processing (see Fig. 9). For the purpose of the tests Spark either passed the values with no modifications or applied average algorithm over 1000 milliseconds period of time.

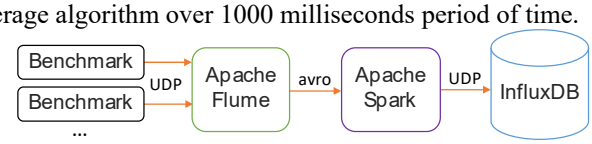

Figure 9: Full Modular stack configuration.

The measured metric rates that the single Spark instance could cope with are:

- 207 kHz in pass through mode.
- 180 kHz in batch processing with average value algorithm.

ISBN: 978-3-95450-193-9 doi:10.18429/JACoW-ICALEPCS2017-TUBPA02

Latency

The latency measurements were executed in Configuration 3 (Benchmark-Flume-InfluxDB). The visualisation was not part of the measurement as real-time updates has not been implemented yet in Grafana (it relies on the metrics from the database). The Figure 10 shows the latency histogram for around 2000 metrics. Most of them were transmitted through the system within 5 µs.

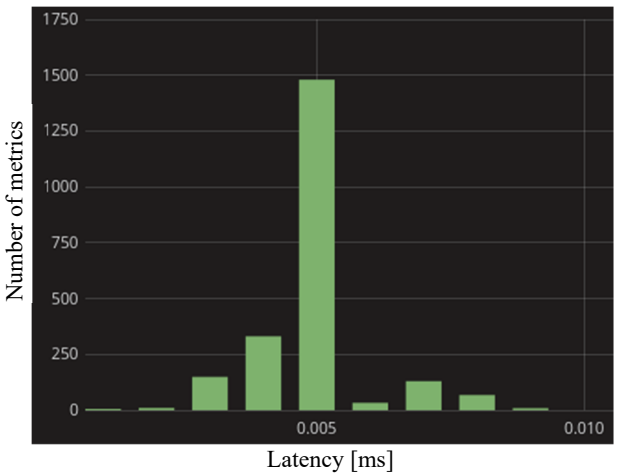

Figure 10: Latency histogram – Benchmark to InfluxDB via Flume.

CONCLUSIONS

The evaluation and results presented in this paper confirm that the Modular Stack is capable of monitoring the future O^2 farm.

The Modular Stack satisfies all functional requirements of metric collection, processing, storage, visualisation and alarming. It is also well documented and supported. It can run in isolation and it is extensively used in other CERN departments and in industry.

The required metric rate of 600 kHz was reached with a single instance of Flume. In addition, a single InfluxDB database was able to store 400 kHz of metrics with an acceptable data loss.

As the monitoring chain uses connectionless UDP protocol, it allows handling large number of sources and imposes low latency (less than 1 ms) from metric generation to the storage.

The major identified risk is a potential need to replace one of the tools. As explained, this risk is acceptable as the migration requires the new component to support clearly identified and standardized protocols.

FUTURE WORK

To complete the Modular Stack latency measurements, additional tests are foreseen to be performed with:

- Visualisation layer.
- Spark with real case processing scenario.

The evaluation and performance tests of Zabbix and MonALISA are ongoing and will be available soon. Once this is done, all considered solutions will be compared and the final selection decision will be taken.

Another future step is the collaboration with other subsystems and detectors to identify processing scenarios and efficiently implement them into the processing unit.

Finally, it is necessary to design an alarming feedback loop that can autonomously take a decision and pass it to the control subsystem when an abnormal but predefined conditions occur.

REFERENCES

[1] ALICE Collaboration, "The ALICE experiment at the CERN LHC", 2008 JINST 3 S08002, 2008.
[2] ALICE Collaboration, "Technical Design Report for the Upgrade of the Online–Offline Computing System", CERN-LHCC-2015-006, 2015.
[3] MonALISA, http://monalisa.caltech.edu/monalisa.htm
[4] Zabbix - The Enterprise-Class Open Source Network Monitoring Solution, https://www.zabbix.com/.
[5] Collectd, https://collectd.org/.
[6] Apache Flume, https://flume.apache.org/.
[7] Apache Flume Developer Guide, https://flume.apache.org/FlumeDeveloperGuide.html
[8] Apache Spark, https://spark.apache.org/.
[9] InfluxDB, https://docs.influxdata.com/influxdb/latest
[10] Grafana - The open platform for analytics and monitoring, https://grafana.com/.
[11] Riemann – A network monitoring system, http://riemann.io/
[12] Unified Monitoring Architecture for IT and Grid Services, https://indico.cern.ch/event/505613/contributions/2227920/attachments/1347026/2041426/Oral-v3-83.pdf
[13] AliceO2Group/Monitoring: The monitoring module for ALICE O2, https://github.com/AliceO2Group/Monitoring
[14] Ansible is Simple IT Automation, https://www.ansible.com/.