EPJ Web of Conferences **214**, 03043 (2019) https://doi.org/10.1051/epjconf/201921403043
CHEP 2018

Towards the integrated ALICE Online-Offline (O^2) monitoring subsystem

Vasco Chibante Barroso[1], *Domenico* Elia[3], *Costin* Grigoras[1], *Andres Gomez* Ramirez[2], *Gioacchino* Vino[3] and *Adam* Wegrzynek[1,*]

[1] European Organization for Nuclear Research (CERN), Geneva, Switzerland
[2] IRI, Goethe University Frankfurt, Frankfurt, Germany
[3] National Institute for Nuclear Physics (INFN), Bari, Italy

Abstract. ALICE (A Large Ion Collider Experiment) is preparing for a major upgrade of the detector, readout and computing systems for LHC Run 3. A new facility called O^2 (Online-Offline) will play a major role in data compression and event processing. To efficiently operate the experiment, we are designing a monitoring subsystem, which will provide a complete overview of the O^2 overall health, detect performance degradation and component failures. The monitoring subsystem will receive and collect up to 600 kHz of performance metrics. It consists of a custom monitoring library and a server-side, distributed software covering five main functional tasks: parameter collection and processing, storage, visualisation and alarms. To select the most appropriate tools for these tasks, we evaluated three options: "Modular Stack", Zabbix and the currently used ALICE Grid monitoring tool called MonALISA. The former one consists of a toolkit including collectd, Apache Flume, Apache Spark, InfluxDB, Grafana and Riemann. This paper describes the monitoring subsystem functional architecture. It goes through a complete evaluation of the three considered options, the selection process, risk assessment and justification for the final decision. The in-depth comparison includes functional features and throughput measurement to ensure the required processing and storage performance.

1 Introduction

1.1 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 LHC (Large Hadron Collider). 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 (from the end of 2009 to 2013) ALICE has been taking data in Run 2 since the beginning of 2015. At

* Corresponding author: adam.wegrzynek@cern.ch

the end of 2018 the LHC will enter into a consolidation phase – the Long Shutdown 2. At that time ALICE will start its upgrade to increase its capacity to collect data by a factor of 100. 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.

1.2 The ALICE O^2 system

The ALICE O^2 computing system [2] will allow the recording of Pb–Pb collisions at a 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 will 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 FLPs (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.

2 The monitoring subsystem

The O^2 monitoring subsystem collects four types of metrics:

- Application – from the O^2 specific software.
- Hardware – from network devices and O^2 specific hardware (such as Common Readout Card [3]).
- Process – performance metrics of each running process.
- System – system performance metrics of each server node.

As presented in Fig. 1, all the metrics are pushed to the processing and aggregation backend for suppression, enrichment, aggregation or correlation. Then, the metrics are written into permanent storage. The storage continuously decreases numerical resolutions of the measured values over time in order to reduce its overall size, e.g. 24 hours after the collection high resolution data is no more needed and each 5-minute block of data points may be replaced with a single point. Stored metrics can be browsed and displayed in the historical record dashboard. In addition, some selected metrics are published to the real-time visualisation and alarming tool in order to allow experts to react to abnormal situations.

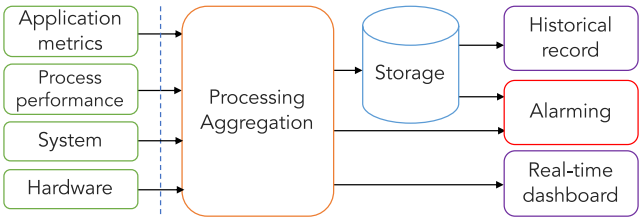

Fig. 1. Metric flow in the O^2 monitoring subsystem.

2.1 Monitoring library

The O^2 monitoring library [4] covers two client-side functional tasks: process monitoring and application metric collection, see Fig. 1. The library can transport values as integers,

floating point numbers and strings. It supports multiple server-side backends. The library gathers process-related metrics such as: uptime, CPU and memory utilisation, bytes sent and received per network interface. It features calculations of derived values such as rate and average. It also appends metrics with metadata (tags) and may send multiple values in a single transaction.

3 Evaluation of tools

The aim of this evaluation is to select the most suitable monitoring tools for the O^2 system. The three preselected options, described in Sec. 3.2, were compared according to the functional and performance requirements.

3.1 Requirements

The monitoring subsystem requirement list was established based on the O^2 Technical Design Report [2]:

1. Compatibility with the O^2 reference operating system (currently CERN CentOS 7).
2. Good documentation.
3. Active maintenance and support by developers.
4. Ability to run in isolation when external services and/or connection to outside of ALICE are not available.
5. Capability of handling 600 kHz input metric rate.
6. Scalability to over 600 kHz if necessary.
7. Handling at least 100 000 sources.
8. Low storage size per measurement.
9. Aligned with the functional architecture introduced in Sec. 2 including system sensors, metric processing, historical record and near-real-time visualisation, alarming and storage supporting downsampling.

3.2 Short list

The list of compared options was initially short-listed to the following: Modular Stack, MonALISA [5], Zabbix [6].

Modular Stack is a set of tools strictly cooperating with each other: collectd [7] as a system performance monitor, Apache Flume [8] as a central router, Apache Spark [9] as a central processor, InfluxDB [10] as a storage, Grafana [11] as a visualisation dashboard and Riemann [12] as an alarming tool.

MonALISA is a complete monitoring framework based on a Dynamic Distributed Service Architecture. It a features self-discovery mechanism, in-memory buffers and supports SQL-like databases.

Zabbix is an open source solution for the real-time monitoring of systems, services and networks. It deploys its own probes, supports a wide variety of storage backends, and provides its own graphical dashboard and management tools.

3.3 Evaluation

Table 1 compares the three options on the functional level – see Sec. 3.1, requirements 1, 2, 3, 4 and 9.

Table 1. Functional comparison of the three options.

	Modular Stack	MonALISA	Zabbix
[1] Reference OS support	Yes	Yes	Yes
[2] Documentation	Good	Insufficient	Good
[3] Support and maintenance	Yes	Yes	Yes
[4] Running in isolation	Yes	Yes	Yes
[9]System sensors	Yes	Yes	Yes
[9]Metric processing	Batch and stream	Stream	Batch
[9]Historical dashboard	Yes	Yes	Yes
[9]Real-time dashboard	No (planned)	Yes (obsolete)	No
[9]Alarming	Yes	Yes	Yes
[9]Storage downsampling	Yes	Yes	Yes

Table 2 presents a performance comparison of the three options - requirements 5, 6, 7 and 8. Multiple test scenarios were completed with various network protocols parameters, CPU and NUMA (Non-Uniform Memory Access) options [13] as they have significant impact in multi-CPU servers. In order to simulate final monitoring traffic as accurate as possible, the O^2 monitoring library was used in the benchmark. The full procedure and detailed results are available in [14].

Table 2. Performance comparison of the three options.

	Modular Stack	MonALISA	Zabbix
[5] 600 kHz metric rate	Yes	Yes	No
[6] Scalable over 600 kHz	Yes	Yes	No
[7] 100k sources	Yes	Yes	No
[8] Storage size	Low	Low	Medium

3.4 Risk assessment

The Modular Stack requires maintaining multiple tools, as well as maintaining compatibility between them. This results in a higher system complexity and the necessity to acquire knowledge about 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 standardised protocols are used for the communication, which can facilitate any future migration. There is also the possibility that newly introduced features will require purchasing a subscription or license.

MonALISA has a low number of components therefore the risk of incompatibility is low. The sender interface is abstracted, as is the database backend so they can be swapped out at any point. With access to the MonALISA internals any changes can be incorporated in efficient manner. However, one component that requires investigating is the Java WebStart real-time display, which should be migrated to a modern web application. MonALISA's core is not open. This is going to change as developers plan to release it on GitHub. Maintenance is assured by Caltech and ALICE Offline. At the moment, the license allows free non-commercial use.

Zabbix is open source, however commercial support requires a license. If a new feature were needed, it would require financial contribution or forking the project and developing custom code on the top of it. Zabbix was built with physical infrastructure and services in mind. It might have problems coping with the huge volume of data that come from monitoring of individual processes. It is also complex with a wide variety of features, which introduces a non-trivial learning curve.

3.5 Selection

After taking into consideration the results it was decided to select Modular Stack for the O^2 monitoring. The decision was justified by the modularity and active maintenance of the tools, the large amount of available resources such us books, tutorials and documentation, variety of system sensors and rich visualisation.

4 Modular Stack implementation

Modular Stack gives a lot of flexibility and each tool can be extended and configured with various parameters. In order to clarify how the final O^2 monitoring subsystem will operate some implementation details are discussed in this section.

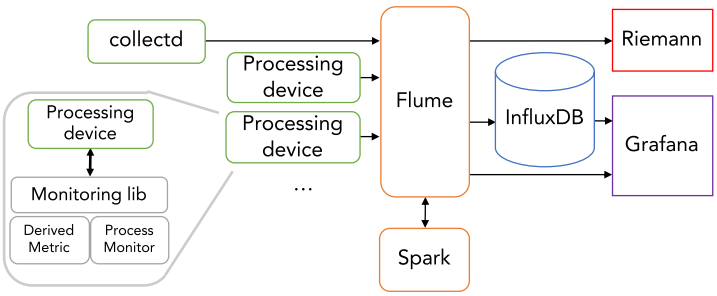

Fig. 2. Modular Stack architecture and metric flow.

4.1 Architecture

As presented in Fig. 2, collectd retrieves system performance metrics (CPU, memory and I/O). It also probes the hardware in order to retrieve its status. These metrics together with values originated from the monitoring library are pushed over the UDP protocol to a server-side tool called Apache Flume. Its main goal is to multiplex metrics between sources and sinks. Flume also performs basic pre-processing (e.g. data suppression and data enrichment) while the more complex computing is delegated to Apache Spark. Spark runs periodical jobs in order to create higher level metrics. As a next step the metrics are pushed to an InfluxDB time-series database. InfluxDB is optimised for high performance writing, imposes low disk occupancy per data point and provides its own data query language. The database engine supports downsampling via Retention Policies and Continuous Queries [15]. The combination of these two features decrease the value resolution over time bringing down the total database size. Grafana serves as a data visualisation tool. Currently it supports rich historical record dashboards; it is already planned to add real-time mode in a near future. It can also generate visual alarms based on values coming from the database. Riemann is used as an alarming tool. It inspects metrics on the fly and generates notifications when abnormal behaviour is detected.

4.2 Metric format

Within the O^2 monitoring subsystem each metric consists of:

- name,
- typed value or values (as integer, double or string),
- timestamp (at least microsecond resolution),
- tags (key-value pairs which behave as metadata).

Metrics coming from the O^2 monitoring library always include hostname, process name and role tags. Multiple values per metric are allowed, e.g. a metric named “memory” may include “user” and “system” values.

4.3 Metric routing

As was mentioned in Sec. 4.1, Flume is responsible for routing the metrics. It deals with a data structure called “Flume event” which consists of a byte array dedicated to storing a value and a key-value vector. Since the Flume event has a different structure than the O^2 metric it was decided to leave the byte array empty and map all the O^2 metric fields to the key-value vector. In addition, all metric values are prefixed with “value_” and all tags with “tag_” literals accordingly.

Flume provides a range of built-in components:

- Source – parses received data into Flume events.
- Sink – converts Flume events into any implemented format.
- Interceptor – component attached to a source that modifies Flume events.
- Memory channel – memory buffer that caches the events.

Flume also provides an API to develop custom components. In order to interface with all the Modular Stack tools, we developed the following components:

- InfluxDB sink - pushes events to InfluxDB via UDP.
- UDP/JSON source – parses JSON encoded metrics sent from the O^2 monitoring library via UDP.
- Collectd JSON handler - reads the data provided by the collectd write_http plugin.

Fig. 3 shows how these components are interfaced.

4.4 Processing and aggregation jobs

The processing jobs run within the Apache Spark environment. Spark executes streaming jobs by splitting the input data stream into batches of data which are processed using the batch functions such as map and reduce. For the moment a single, generic job was defined which covers most of the aggregation cases (aggregating per detector, per machine, etc.). It operates on Flume events therefore no conversion is required. It is configurable – per each metric the following configuration parameters can be set:

- Processing function (average, sum, minimum, maximum).
- Aggregation per tag (e.g. per detector name, per all detectors, per same machine, etc.).
- Time window.

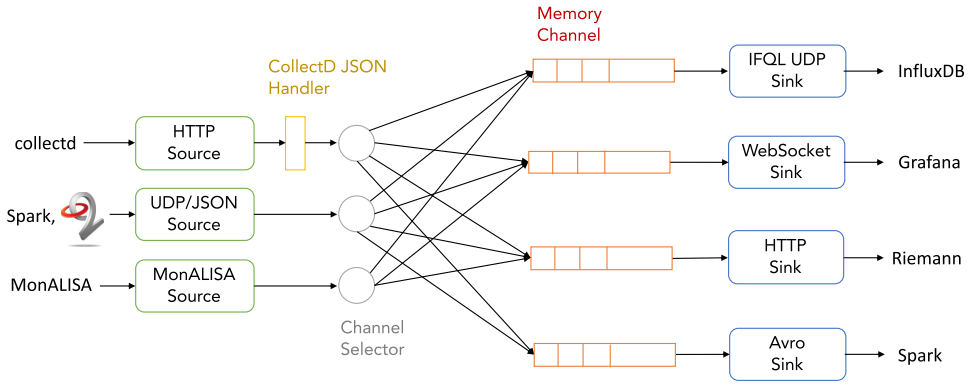

Fig. 3. Flume internal architecture.

4.4 Integration with the O^2 software

The O^2 Monitoring subsystem has already been adapted to work with other O^2 components in order to visualise metrics crucial for the commissioning process. These components use the monitoring library in order to inject values into the subsystem. The following activities are monitored:

- Quality Control [16] – publishing data quality histograms.
- Data Processing Layer [17] – status of data processing topologies.
- Readout [18] – data transfer from the FPGA cards into a computer's memory.

5 Conclusion

After an extensive evaluation with functional and performance tests of the three options (Modular Stack, MonALISA and Zabbix) it was decided to select the Modular Stack for the O^2 monitoring. Currently, the project is in the implementation phase with a number of final details still being clarified and the custom components developed. Recently, several test setups were deployed in order to verify the subsystem behaviour in a production environment as early as possible. These setups also deliver status dashboards for detector commissioning and hardware performance tests, thus providing an added value to developers of other components.

References

1. ALICE Collaboration, *The ALICE experiment at the CERN LHC,* 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. J. Mitra et al, *Common Readout Unit (CRU) - A new readout architecture for the ALICE experiment*, JINST **11** C03021 (2016)
4. *ALICE O^2 monitoring library*, https://github.com/AliceO2Group/Monitoring, accessed 2018-10-10
5. *MonALISA*, http://monalisa.caltech.edu, accessed: 2018-10-10
6. *Zabbix The Enterprise Class Open Source Network Monitoring Solution*, https://www.zabbix.com, accessed 2018-10-16
7. *Collectd – The system statistics collection daemon*, https://collectd.org, accessed 2018 - 10-10
8. *Apache Flume*, https://flume.apache.org, accessed 2018-10-16
9. *Apache Spark - Unified Analytics Engine for Big Data*, http://spark.apache.org, accessed 2018-10-16
10. *InfluxDB - The Time Series Database in the TICK Stack*, https://www.influxdata.com/time-series-platform/influxdb/, accessed : 2018-10-08
11. Grafana - *The open platform for analytics and monitoring*, https://grafana.com, accessed 2018-10-16
12. *Riemann - A network monitoring system*, http://riemann.io, accessed 2018-10-16
13. Christopher Hollowell et al, *The Effect of NUMA Tunings on CPU Performance*, J. Phys.: Conf. Ser. **664** 092010 (2015)
14. V. Barroso, G. Vino, A. Wegrzynek, *Monitoring the New ALICE Online Offline Computing System*, ICALEPCS'17, pp. 195-200 (2018)
15. *InfluxDB – Downsampling and data retention*, https://docs.influxdata.com/influxdb/v1.6/guides/downsampling_and_retention/, accessed 2018-10-16
16. B. von Haller, P. Lesiak, J. Otwinowski, *Design of the data quality control system for the ALICE O^2*, J. Phys. Conf. Ser. **898** 032001 (2017)
17. G. Eulisse, P. Konopka, M. Krzewicki, M. Richter, D. Rohr, S. Wenzel, *Evolution of the ALICE Software Framework for LHC Run 3*, these proceedings (2019)
18. F. Costa, S. Chapeland, *Readout software for the ALICE integrated Online-Offline (O2) system*, these proceedings (2019)