ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

A MONITORING SYSTEM FOR THE NEW ALICE O2 FARM

G. Vino†, D. Elia, INFN, Bari, Italy
V. Chibante Barroso, A. Wegrzynek, CERN, Geneva, Switzerland

Abstract

The ALICE Experiment has been designed to study the physics of strongly interacting matter with heavy-ion collisions at the CERN LHC. A major upgrade of the detector and computing model (O2, Offline-Online) is currently ongoing. The ALICE O2 farm will consist of almost 1000 nodes enabled to readout and process on-the-fly about 27 Tb/s of raw data.

To increase the efficiency of computing farm operations a general-purpose near real-time monitoring system has been developed: it lays on features like high-performance, high-availability, modularity, and open source. The core component (Apache Kafka) ensures high throughput, data pipelines, and fault-tolerant services. Additional monitoring functionality is based on Telegraf as metric collector, Apache Spark for complex aggregation, InfluxDB as time-series database, and Grafana as visualization tool.

A logging service based on Elasticsearch stack is also included. The designed system handles metrics coming from operating system, network, custom hardware, and in-house software. A prototype version is currently running at CERN and has been also successfully deployed by the ReCaS Datacenter at INFN Bari for both monitoring and logging.

INTRODUCTION

The ALICE Experiment

ALICE (A Large Ion Collider Experiment) [1] is a detector designed to study the physics of strongly interacting matter (the Quark–Gluon Plasma), produced in heavy-ion collisions 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 the successful Run 1 (2010-2013) and Run 2 (2015-2018) data taking periods, the LHC entered into a consolidation phase (Long Shutdown 2) and ALICE started its upgrade to fully exploit the increase in luminosity expected in Run 3. The upgrade foresees a complete replacement of the computing systems (Data Acquisition, High-Level Trigger and Offline) by a single, common O2 (Online-Offline) system.

The ALICE O2 System

The ALICE O2 computing system [2] will allow the recording of Pb–Pb collisions at a 50 kHz interaction rate.

† gioacchino.vino@cern.ch

Some detectors will be read out continuously, without physics triggers. Instead of rejecting events the O2 system will compress the data using 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 O2 facility will consist of 200 FLPs and 750 EPNs. The O2 farm will receive data from the detectors at 27 Tb/s, which after FLP and EPN processing will be reduced to 720 Gb/s.

MONITORING SYSTEM OBJECTIVES

The Monitoring subsystem is part of O2 and provides comprehensive functionality in metric collection, routing, processing, storage, visualization and alarming as shown in Fig. 1.

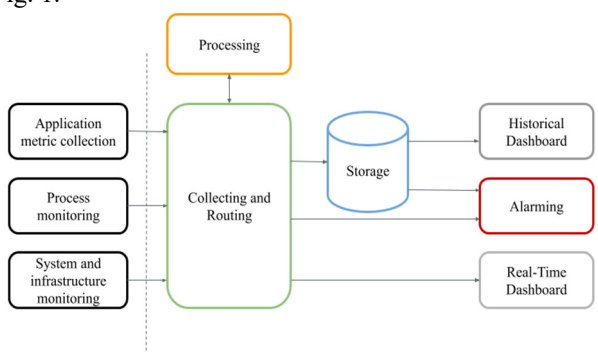

Figure 1: Functional architecture of the system.

Three classes of metrics (application, process and system/infrastructure) are collected and pushed to the Collecting and Routing backend. Metrics requiring processing are forwarded to the Processing component that injects back the processed values. Then, all the values are written into permanent storage. From that point they can be browsed and visualized in the historical record dashboard. Selected metrics are published for alarming and real-time visualization.

System and Infrastructure Monitoring

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

- CPU
- Memory
- Network
- Storage
- Hardware status

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

It can also query devices such as network switches, routers and power supplies via standardized or generally available protocols (SNMP, IPMI) 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 O2 process such as:

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

It is implemented via a library linked to the process.

Application Metric Collection

The Application metric collection provides an entry point from O2 processes to the Monitoring subsystem. It forwards user defined metrics to the processing backend via connection or connectionless transport protocols.

Collecting and Routing

This component collects all monitoring data coming from all sources and forwards them toward selected backends like storage, real-time dashboard, alarming or to the Processing component.

Processing

The Processing correlates and manipulates metrics coming from different origins. It may occur at any step of the monitoring chain, including the central collector if correlations between widely different metrics are needed. Processing is the only component that reads data from and writes to the Collecting and Routing component. The processing task types are:

- 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).

Storage

The Storage receives and writes metrics into a historical record. It must support large input metric rates. It accepts queries to retrieve stored metrics. It also provides streams of messages administration tools to manage its internal configuration. Given that the O2 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 the 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 O2 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 1000 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 MONITORING SYSTEM OBJECTIVES section.

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

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

ARCHITECTURE

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

Metric Collection relies on two components: Telegraf [3] (which replaces initially selected CollectD [4] as it introduces strings support) and a custom C++ monitoring library [5] providing convenient interface in order to be used within different parts of O2 system. Telegraf ships a large set of plugins able to collect heterogeneous metrics from

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

the hardware and operating system (CPU, memory, network, SNMP and IPMI). It outputs metrics in a compact format called InfluxDB Line Protocol [6]. This protocol uses labels (aka tags) to better identify carried values. As Telegraf covers the system and infrastructure the O2 monitoring library reports process performance metrics and passes user-generated values to a selected backend. It has been already integrated in multiple parts of the O2 software [7]: Data Processing Layer (DPL), Quality Control and Readout.

Initially, a large number of monitoring data producers was foreseen (as each of the 100k processes expected in O2 at the time of the TDR would have created a monitoring channel over the library) and thus the Collecting and Routing component was supposed to manage a large number of connections. This number was reduced by having Telegraf, as a local metric collector on each node, gathering the values from all monitoring library instances over a Unix socket. This approach decreases the number of connections from 100k to about 1000.

In the first version, Collecting and Routing was covered by Apache Flume [8] but, as it does not provide fault-tolerance and scalability, it was decided to replace it with Apache Kafka [9]. Since this new solution includes a module (Kafka Streams [10]) that could be easily used as processing component, it was decided to remove Apache Spark [11] from the initial stack. Most of the Kafka features (e.g. scalability, fault-tolerance and data pipelines) are accomplished thanks to the concept of topic: a stream of messages sharded into partitions. These partitions are replicated and distributed for high availability into Kafka servers (aka brokers). Scalability is achieved by partition configuration, increasing the topic partitions leads to higher throughput but at the expense of increased latency [12]. The fault-tolerance is controlled by a replication factor: the higher the replication factor is the more broker failures can be tolerated. Increasing the replication factor leads to significantly higher I/O usage, therefore it's crucial to estimate the optimal partition and replication factor value that fit the target performance. Some Kafka features like pull-based consumers, writing all data to the disks and complex protocol don't fit our use-case but, as described in the next section, this will not impact on the final system performance. Kafka uses Apache Zookeeper [13] to manage brokers, topics and partitions dynamically and with high reliability.

Processing tasks described in the MONITORING SYSTEM OBJECTIVES have been implemented using Kafka Streams that inherits scalability and fault-tolerance features from Kafka. Average, sum, minimum and maximum are the currently available aggregation functions.

Since all monitoring data can be classified as time-series, InfluxDB [14], a time-series database, has been selected as storage. It features high performing writing, low disk occupancy and an optimised querying. The InfluxDB engine supports Retention Policy and downsampling via Continuous Queries to limit disk usage. A custom InfluxDB Kafka consumer has been implemented to send the monitoring data from Kafka to the database over UDP.

Grafana [15] has been chosen as the data visualisation tool. It supports both real-time and historical-record dashboards. It can also generate alarms based on values coming from the database and forward notifications to outside systems.

A dedicated notification service has been created in order to handle the alarm notifications from Grafana and those generated in real-time during the processing. The service receives notifications either as HTTP requests or over Kafka protocol and passes them to email and Mattermost channels. Figure 2 shows the architecture of the described system.

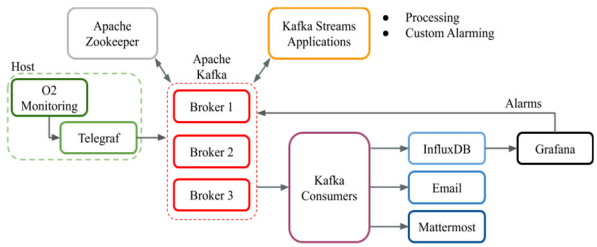

Figure 2: Architecture of the monitoring system.

SYSTEM PERFORMANCE TESTS

Test Description

The test aims to measure to maximum performance of the system, find bottlenecks when possible and compare them with requirements.

The first test verifies Kafka's performance and analyzes bottleneck, especially:

- Disk writes
- Number of connections between brokers and producers.
- Writing metrics from consumers to InfluxDB.

Then the following tests measure:

- Latency and percentage of lost messages at 600kHz metrics coming from 1000 producers.
- Performance of aggregation task.
- Ability to manage a broker failure.

Testbed Description

The testbed is composed of:

- 3-Kafka broker machines:
  - 128 GB RAM.
  - 25 Gigabit Ethernet.
  - 100 MB/s Disk I/O (single HDD disk).
  - 32 CPU cores.
- 1 InfluxDB server:
  - 128 GB RAM.
  - 25 Gigabit Ethernet.
  - 600 MB/s Disk I/O (single SSD disk).
  - 32 CPU cores.
- And the remaining machines with commodity server hardware and 1 Gbps connection:
  - 1 machine for Zookeeper service.
  - 3 machines as consumers (writing data from Kafka to InfluxDB).

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

- 5 machines as producers (writing data to Kafka).

As atomic monitoring data a message with a 100 bytes length has been composed since it represents the worst case of a single value per message with three tags. Figure 3 shows the architecture used for the benchmarks.

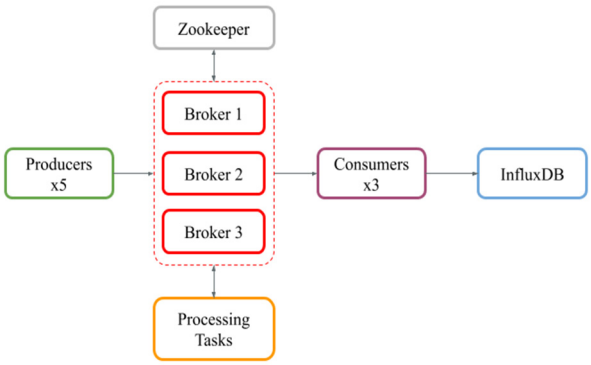

Figure 3: Architecture used for benchmarks.

The number of topic partitions is 6 (2 partitions per broker). Besides, the topic replication factor is 2 which allows to tolerate a single broker failure at the cost of doubling the network and disk I/O.

Test Procedure

The test procedure is semi-automatized thus allowing for a test to be quickly repeated in any of the configurations. The benchmark is based on the O2 Monitoring library and can be deployed and controlled via Ansible [16]. Kafka statistics are exposed using Jolokia [17] and collected over HTTP using Telegraf. Latency was measured by passing a metric through the system and inserting a timestamp at each step. A custom script extracts statistical information from these values. Clock synchronization is handled with Network Time Protocol (NTP) daemon [18]: tests demonstrated sub-millisecond clock mismatch among machines which is acceptable for the goal of this test.

Max Input Metric Rate vs. Writes on Disks

The test was executed using a single disk per broker. 1000 producer instances (Monitoring Library over librdkafka [19) were used. Results are promising as the measured maximum input metric rate was 1.5 M msg/s and 150MB/s in total or 500k msg/s and 50MB/s per broker. Considering the replication factor of 2, the actual write speeds on disk was 100 MB/s. In this case, the disk is the bottleneck, but the measured maximum input metric rate was significantly higher than the 600k kHz required. A higher maximum input rate could be obtained moving to higher I/O disks, adding further disks or brokers.

Max Number of Connections vs. Complex Protocol

This test measures the impact of large numbers of established TCP connections between producers and brokers by increasing the number of producer instances until the limit is reached. The measured max number of TCP connections is above than 50k for the whole cluster. Using a 6-partition topic, each producer is connected to the 3 brokers with 4 TCP connections. This means the maximum number of producers that can be managed by 3 Kafka brokers is around 12k, compared to 1000 in the requirement.

Maximum Writing Rate in InfluxDB

The test aims to measure the maximum writing rate into InfluxDB, which depends on the database itself but also on Kafka consumer configuration. A consumer is a custom component that retrieves metrics from brokers and passes them over UDP to the InfluxDB instance. The UDP protocol has been used (instead of default HTTP) as it is more performant. Table 1 shows the receiving rate as a function of number of consumer instances and the number of UDP ports.

Table 1: InfluxDB Writing Rate in ksample/s as a Function of Number of Consumer and Number of Ports

#Ports	1 consumer	2 consumers	3 consumers
1	220	380	375
2	310	440	540
3	290	510	630
4	284	520	760
5	270	575	730
6	276	560	750

A single InfluxDB instance can store all 600 kHz metric rate. The maximum writing rate of a single consumer instance is around 250-300 kHz, so 3 instances could cover the requirement. Although for fault-tolerant purpose it is recommended to have 1 spare instance as in case of failure 2 instances will not be able to cope with the load. The number of 6 consumers might be an optimal solution since there are 6 partitions in use.

Aggregation Task Performance

The aggregation tasks are custom components relying on Kafka Streams library. Four aggregation functions have been implemented: average, sum, minimum and maximum. The components compute the aggregated value over selected metrics and over a configurable time window. Results are written into a dedicated topic using InfluxDB line protocol. The test aims to measure the maximum rate of each processing function. The measured value was nearly the same for all the functions and corresponds to 250k metrics/s. In the case a higher value is required, the number of aggregation instances could be increased.

Ability of System to Tolerate a Broker Crash

This test aims to verify whether the system could manage a failure. Brokers are core elements and their performance affects the overall system. This test evaluates whether the system works with only 2 out of 3 active brokers. Figure 4 shows the capability of the system to manage the failure of a single broker by splitting the traffic among the remaining active brokers and restoring the normal state

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

after a short period of time. This scenario simulates the restart phase of the broker by the O2 control system.

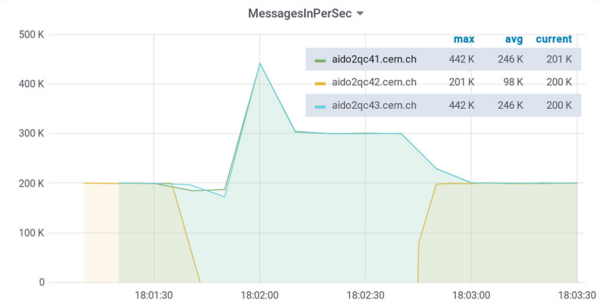

Figure 4: Input metric rate per broker as a function of active brokers.

Latency and Message Lost

This test aims to measure the latency between the metric generation phase and the metric storing phase. The maximum allowed value is 1000ms to the visualisation layer and 500ms to the processing layer. Messages are sent from 4 machines. Latency is measured as the timestamp difference between the moment the metric was created (producer) and stored (InfluxDB). Figures 5-7 show the latency as a function of metric rate for 1, 100 and 1000 producers, respectively. On the horizontal axis the percentile of handled metrics is displayed. As expected, latency increases with number of producers and metric rate.

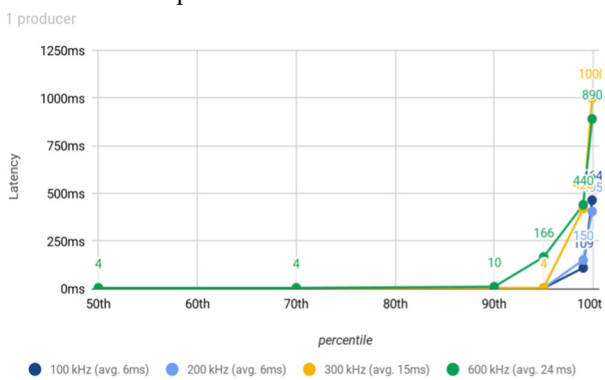

Figure 5: Latency using a single producer.

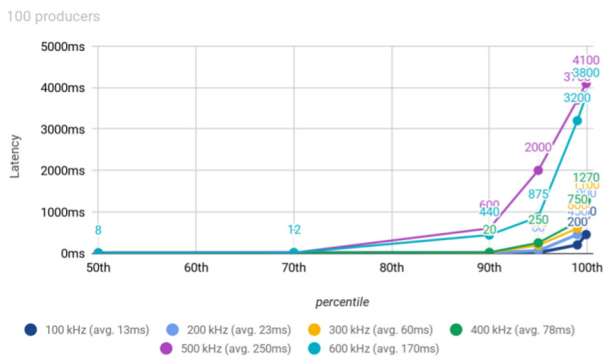

Figure 6: Latency using 100 producers.

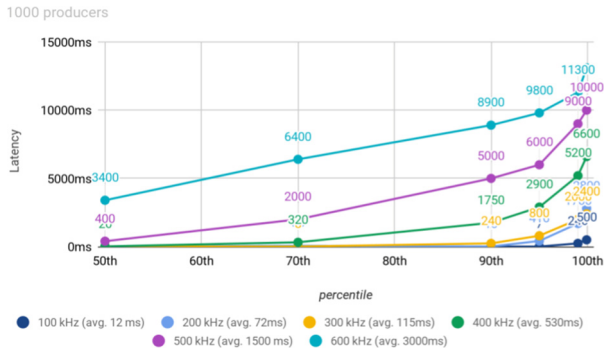

Figure 7: Latency using 1000 producers.

Figure 7 represents the worst-case scenario where all Telegraf instances send data without local aggregation and Kafka does not process any values. It is foreseen that processing will significantly reduce the rate from 600 kHz input rate, with less than half being written to the database. In each test the percentage of lost messages never reached over 0.1%.

CONCLUSIONS

The results presented in this paper confirm that the O2 Monitoring system design and preliminarily implemented custom components are capable of handling the monitoring traffic from the future ALICE O2 farm. The system also satisfies all functional requirements of metric collection, processing, storage, visualisation and alarming. Apache Kafka was able to collect, route and process all incoming metrics in a scalable and fault-tolerant way. In addition, a single InfluxDB database could store 600 kHz of metrics.

Latency test showed our Kafka cluster is able to collect at least 300 kHz metrics from 1000 producers with a latency lower than 500 ms. We plan to improve this value by decreasing the Kafka input rate via local aggregation in the Telegraf instances running on each node. Moreover, adding more performant disks and Kafka brokers will drop the latency further.

The presented system is generic enough to carry all types of messages (not only metrics). For conformation, a logging service was implemented and successfully deployed by the ReCaS Datacenter at INFN Bari.

The current implementation of the system is almost complete but additional advanced features are foreseen to be added. More complex processing and alarming tasks, using correlation, derived values, multiple thresholds and machine learning algorithms, might be needed.

The near future challenge is the collaboration with other subsystems and detectors to identify processing scenarios and efficiently implement them into the processing unit. Finally, an alarming feedback loop needs to be added to the system in order to autonomously adjust the system by passing signals to the O2 Control [20] system when an abnormal or predefined condition occurs.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUDPP01

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] Telegraf, https://www.influxdata.com/time-series-platform/telegraf/

[4] Collectd, https://collectd.org/

[5] AliceO2Group/Monitoring: The monitoring module for ALICE O2, https://github.com/AliceO2Group/Monitoring

[6] InfluxDB Line protocol, https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_tutorial/

[7] O2 software, https://github.com/AliceO2Group

[8] Apache Flume, https://flume.apache.org/

[9] Apache Kafka, https://kafka.apache.org/

[10] Apache Kafka Streams, https://kafka.apache.org/documentation/streams/

[11] Apache Spark, https://spark.apache.org/

[12] Connecting to Apache Kafka, https://www.confluent.io/blog/how-choose-number-topics-partitions-kafka-cluster

[13] Apache Zookeeper, https://zookeeper.apache.org/

[14] InfluxDB, https://docs.influxdata.com/influxdb/latest

[15] Grafana - The open platform for analytics and monitoring, https://grafana.com/

[16] Ansible is Simple IT Automation, https://www.ansible.com/

[17] Jolokia, https://jolokia.org/

[18] Network Time Protocol (NTP), http://www.ntp.org/

[19] librdkafka, https://github.com/edenhill/librdkafka

[20] O2 Control, https://github.com/AliceO2Group/Control