A web-based solution to visualize operational monitoring data in the Trigger and Data Acquisition system of the ATLAS experiment at the LHC

G Avolio[1], M D'Ascanio[1], G Lehmann-Miotto[1] and I Soloviev[2]

[1] European Laboratory for Particle Physics, CERN, Geneva 23, CH-1211, Switzerland
[2] University of California, Irvine, CA 92697, USA

E-mail: Giuseppe.Avolio@cern.ch and Igor.Soloviev@cern.ch

Abstract. The Trigger and Data Acquisition (TDAQ) system of the ATLAS detector at the Large Hadron Collider at CERN is composed of a large number of distributed hardware and software components (about 3000 computers and more than 25000 applications) which, in a coordinated manner, provide the data-taking functionality of the overall system. During data taking runs, a huge flow of operational data is produced in order to constantly monitor the system and allow proper detection of anomalies or misbehaviours. In the ATLAS trigger and data acquisition system, operational data are archived and made available to applications by the P-BEAST (Persistent Back-End for the Atlas Information System of TDAQ) service, implementing a custom time-series database. The possibility to efficiently visualize both real-time and historical operational data is a great asset facilitating both online identification of problems and post-mortem analysis. This paper will present a web-based solution developed to achieve such a goal: the solution leverages the flexibility of the P-BEAST archiver to retrieve data, and exploits the versatility of the Grafana dashboard builder to offer a very rich user experience. Additionally, particular attention will be given to the way some technical challenges (like the efficient visualization of a huge amount of data and the integration of the P-BEAST data source in Grafana) have been faced and solved.

ATL-DAQ-PROC-2017-001
02/05/2017

1. Introduction

The Information Service (IS) [1] is a software infrastructure component of the Trigger and Data Acquisition system (TDAQ) [2] of the ATLAS [3] detector at the Large Hadron Collider (LHC) at CERN. IS is used for the online monitoring of both the TDAQ system and the ATLAS sub-detectors. It is implemented as a set of CORBA [4] server applications running on dedicated computers at the ATLAS experimental area. The TDAQ and detector applications use this service to publish information objects of various types, to read them and to subscribe to updates of the objects to be notified of changes. The experts use interactive graphical tools to monitor the state of ATLAS systems and to diagnose possible problems online.

There are cases when the experts would like to analyze the history of IS data in order to better understand the behavior of the system in the past. However, IS does not provide persistent data storage, so if the service is not running any more or information has been updated, the previous IS data are lost and not accessible to experts. For these reasons there was a request to add generic persistence for IS data and to provide facilities for their retrieval and analysis.

2. User Requirements and Design

The high-level functionality provided by this project is a generic mechanism to store, retrieve and visualize ATLAS operational monitoring data. There are two important functional requirements that must be supported: the structure of the data may evolve in time and the stored information should not lose its granularity over time. From the performance and scalability point of view the rates of monitoring data updates are rather high and reach an order of 100,000 updates per second for the entire ATLAS experiment. In addition, the system should work behind the scenes, in no way influencing the operation of the data-acquisition system. Moreover there are restrictions on available hardware resources for implementation at the experimental site (known as Point-1) in terms of number of computers, disk space and network bandwidth. Thus it has to be possible to move the oldest or least used data outside Point-1 to a long term storage location with larger capacity.

The main users are experts analyzing behavior of different systems of the ATLAS detector, investigating problems, correlating data and comparing differences between data taking sessions. Another group of users are shifters using online dashboards to verify the state of various parameters of the experiment during on-going runs. Additionally, operational data can serve as a resource for reports, presentations or papers. The user can request data correlation and aggregation prior to visualization.

The high level design (figure 1) includes several distinct components. The *receiver* component gathers operational monitoring data from the ATLAS online system at Point-1 and puts them into distributed *persistent storage*. The *data access* component provides remote access to the persistent storage. Data can be accessed via programming interfaces, command line utilities and graphical web-based applications.

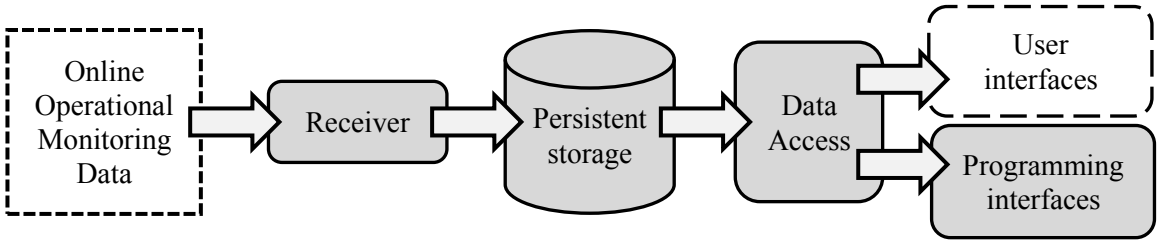

Figure 1. The high level design of the operational monitoring data persistency and visualization service.

For efficiency reasons the receiver aggregates and buffers data before moving them to storage. Therefore, to support quasi real-time data access, the receiver can also be queried directly by the data-access component. This path is not shown in figure 1.

3. Data Persistency Service Implementation

The data persistency service is responsible for gathering operational monitoring data, putting them into persistent storage and providing programming interfaces to them.

3.1. The First Prototype

The first prototype for persistent IS data storage was developed and deployed a few months before the end of LHC Run 1. It was named Persistent Back-End for the Atlas Information System of TDAQ (P-BEAST) [5]. For that first prototype of P-BEAST a modern, distributed, Java-based Cassandra database [6] was used as the online storage technology, while the CERN EOS [7] was chosen for long-term archival storage.

Despite the use of three clustered servers and the selection of a reduced data sample combined with smoothing, the Cassandra implementation was never fast enough to keep up with the rate of incoming data. Furthermore, disk space was used inefficiently, with only 20% of the volume dedicated to actual operational data and the rest reserved for Cassandra database maintenance and information replication. As a consequence, the prototype used Cassandra only as a temporary store for about one month of

information. Older data were moved to EOS as compressed JSON files. However, an effective way of querying them back was not implemented. Finally, the prototype did not support data arrays and nested data structures nor redefinition of the data classes.

3.2. *Present Implementation*

Based on the experience with the Cassandra database, it was decided to redesign P-BEAST using a different approach. The new system uses the same database technology for both the online storage at Point-1 and the long-term storage on EOS. It stores data into an efficient custom binary format with random data access based on Google protocol buffers [8] encoding for data compactness, high performance, cross-platform and programming language interoperability.

The first functional P-BEAST version was implemented and deployed by the end of 2013 and improved during the course of 2014 [9]. The new P-BEAST is able to archive all data types supported by IS including primitive types, arrays and nested types, and is able to take into account data deletion (i.e. closing interval of data validity). It provides a C++ interface for direct access to data files, a CORBA interface for remote access and a REST interface for web-oriented applications. An application can choose to read either raw or down-sampled data. Quasi real-time access to just-published IS data is also provided at Point-1. Once more and more experts started to use P-BEAST data for analysis, a requirement to implement more convenient C++ and Python interfaces working within and outside Point-1 was introduced. These two interfaces were added to P-BEAST in the 3rd quarter of 2016.

The P-BEAST service is implemented by several cooperating applications running on dedicated computers. The *receiver* application is used to gather monitoring data from the IS servers, buffer and process them in memory removing duplicated values (data compaction), reorder IS callback data chronologically, and finally save all data to P-BEAST files on a local file system. In addition, the receiver implements a CORBA interface providing client access to data not yet offloaded to persistent storage. Another application is the *merger* that periodically checks the state of the P-BEAST repository and merges many small data files above a certain age produced by the receiver into large files. Such large files are stored in the central Point-1 repository and then copied to EOS for long-term storage. The *server* application provides the CORBA interface to the P-BEAST data files, and, if necessary, performs in-memory or persistent down-sampling. The persistently down-sampled data are stored into a cache repository (with *a discard the least recently used items first* strategy) and the original raw data remain unmodified. Finally, the *P-BEAST server* application accepts user requests, communicates with receivers and servers, merges answers obtained from them and returns the result to clients. Alternatively, clients can access data from EOS using programming interfaces. Figure 2 gives an overview of the P-BEAST architecture.

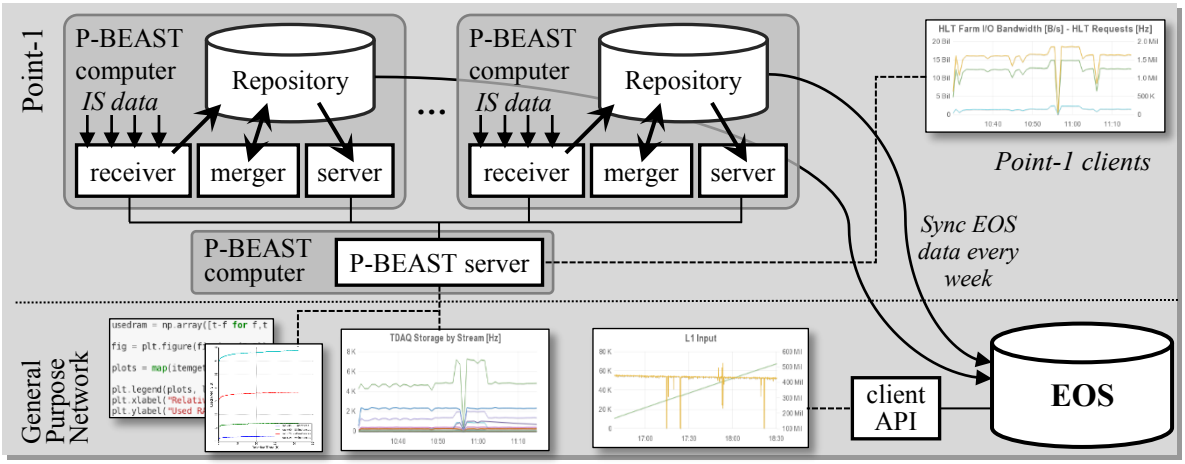

Figure 2. P-BEAST architecture.

The service is made scalable horizontally and vertically. There is at least one receiver process per data acquisition instance (partition). A receiver can use all available CPU cores and RAM resources. If the performance of a single computer is not enough to sustain the information processing rate, the design of the service makes it possible to run several receivers per partition on different computers to split the load. A dedicated disk-balancing mechanism moves data across storage servers to equalize their disk occupancies. In the middle of 2015 two new P-BEAST computers with dual 12 core CPUs, 256 GB RAM and 8x4 TB RAID arrays were commissioned. The IS update rates from all partitions during data taking runs are several times below the capabilities of either new machine measured during recent performance tests, and the total disk space is enough to store raw IS data archives at Point-1 for a minimum of three years.

The main clients of the P-BEAST system are graphical interactive web-based monitoring data dashboards, which will be discussed in detail in the next section.

There are plans to directly use P-BEAST data in CERN's Service for Web based Analysis (SWAN) [10] allowing complex data correlations. For the moment the raw P-BEAST data are converted into CSV format being imported into the SWAN environment.

Also, the data stored using P-BEAST are used by the ATLAS Shifter Assistant Replay (SAReplay) [11] service. The SAReplay is able to reconstruct IS objects from P-BEAST data for offline simulation of real data taking run conditions.

4. Data visualization

The capability to properly and efficiently visualize operational data is of crucial importance in a system as complex as ATLAS data acquisition. With its capacity to archive and serve clients with operational data in quasi real-time, the P-BEAST service presents itself as a natural data source for the data visualization system.

The developed web solution is based on the definition of *dashboards*, collecting and reporting data in different domains (as an example, a domain may be represented by a specific ATLAS sub-detector). The dashboard approach enables the efficient organization, segmentation and grouping of the available data-set, offering the final user a complete view on critical system parameters.

4.1. The dashboard manager

Dashboards are managed by the *Grafana* [12] dashboard builder. *Grafana* is an open source project for visualizing metrics as time series in a web browser, mainly based on the AngularJS [13] framework. It makes it possible to create dashboards with fully interactive and editable graphs, gathering data from different back-ends. *Grafana* has a pluggable design, allowing users to extend its functionality in order to support custom data sources. Using this feature it was possible to seamlessly integrate a purpose-built P-BEAST back-end.

Grafana natively supports the definition of time ranges for the data being visualized, offering a single unique solution for both real-time and historical data visualization.

4.2. P-BEAST integration

The integration of P-BEAST into the *Grafana* framework was the main challenge faced during the development phase. The integration has included:

- The implementation of a P-BEAST REST interface, exposing methods to transfer both data and meta-data to the *Grafana* client;
- A *data formatter* module (on the client), post-processing data in order to make them compatible with the *Grafana* model;
- User interface elements to configure and set P-BEAST specific query variables.

Grafana uses the JSON format to exchange data with its back-ends. Particular attention has been paid to the definition of the data structure representing the P-BEAST data. Given the potentially huge

amount of data to be shown, a too-expressive JSON format has proven to be a bottleneck in performance. The *data formatter* was therefore introduced with the goal of having the smallest data transfer over the network with the smallest amount of post-processing in the web browser.

4.3. Performance

The main performance metric for the data visualization component is the responsiveness. Ideally, dashboards with many plots should require minimal update time.

In the case of *Grafana*, figure 3 shows the time needed by the browser to render a dashboard with a single graph as a function of the number of drawn time series and for different number of data points per series. Data are internally generated, so that the results do not depend on the performance of the P-BEAST back-end (including the network latency), but just on the *Grafana* software stack and the browser's JavaScript engine. The tests were performed using *Firefox* ESR 45.3.0 and *Grafana* 1.9.1**,** on a desktop-like PC equipped with an Intel i7-3770 CPU [14] and running the Scientific Linux 6 [15] operating system. As can be seen, the rendering time can be kept under control tuning the number of data points per time series, so that a graph can be made up of several hundreds of time series.

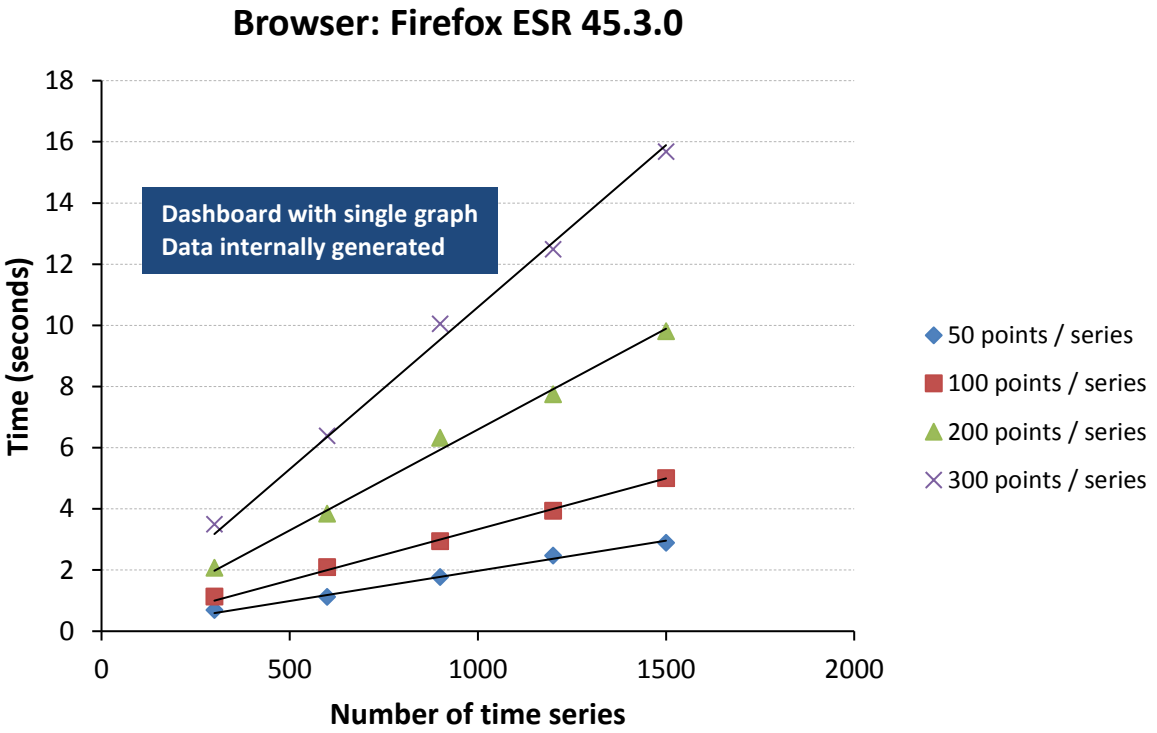

Figure 3. Browser rendering times for a dashboard with a single graph, as a function of the number of time series and the number of data points per time series.

Figure 4 shows the typical interaction times between the *Grafana* dashboard and the P-BEAST back-end. Tests have been performed with a single graph with 600 time series and varying the number of data points per series, using the same browser and *Grafana* version previously mentioned. Reported times are:

- *Waiting*: time waiting for a response from P-BEAST;
- *Receiving*: time taken to read the entire response from P-BEAST;
- *Processing*: time taken by the *data formatter* module to properly format data received from P-BEAST and inject them into the *Grafana* engine.

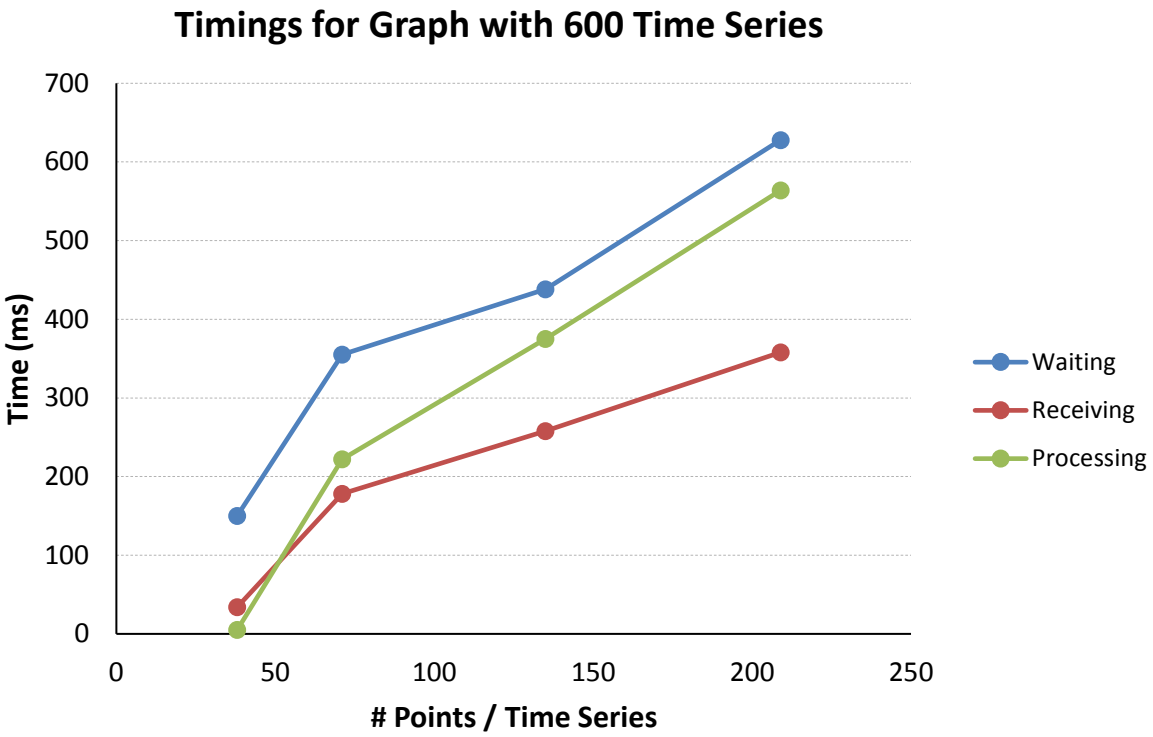

Figure 4. Timings for a dashboard with a single graph and 600 time series as a function of the number of data points per time series.

As can be seen, the processing time is very low for time series with about 50 data points and reaches about 560 ms for 209 data points per time series.

Taking into account the results of the performance tests, three fundamental design choices have been made:

- Parametrize the queries to P-BEAST specifying a maximum number of data points to be returned per time series (the P-BEAST service takes care of applying a proper *down-sampling* to the data);
- Data points in each time bin are augmented with a maximum and minimum value in order to enhance series visualization with *validity bands*. In this way it is possible to properly follow data trends and eventually identify peaks and dips even when the number of data points in the series is set to a very low value;
- Favour a slim JSON data description at the expense of some processing time spent by the *data formatter* module.

In that way the user is given the possibility to configure rich and complex dashboards keeping performance well under control.

4.4. Dashboard deployment

Dashboards are described as JSON files and can be created using the web user interface (UI), without any need of manual editing. The UI makes it possible to define the content, layout and plotting options of all the graphs in the dashboard. It is additionally possible to set an automatic refresh rate or to select a specific time interval for data retrieval. Some UI elements have been customized to work with the P-BEAST back-end and to enable special options:

- Guided query-builder with auto-completion;
- Definition of the maximum number of data points per time series;
- Enabling/disabling of data *validity bands*.

Grafana 1.9.1 is a client-only web application. Critical dashboards are uploaded to a protected web server area and made available to the ATLAS community. At the same time, users can create *personal* dashboards, save them locally and upload to *Grafana* via the browser when needed.

Currently about 20 different dashboards are configured and regularly consulted by experts and shifters in the ATLAS control room. The richest dashboards are made up of more than 20 plots with single graphs showing up to 1000 time series.

Figure 5 shows a dashboard reporting in quasi real-time the status of the P-BEAST service during typical ATLAS data-taking sessions.

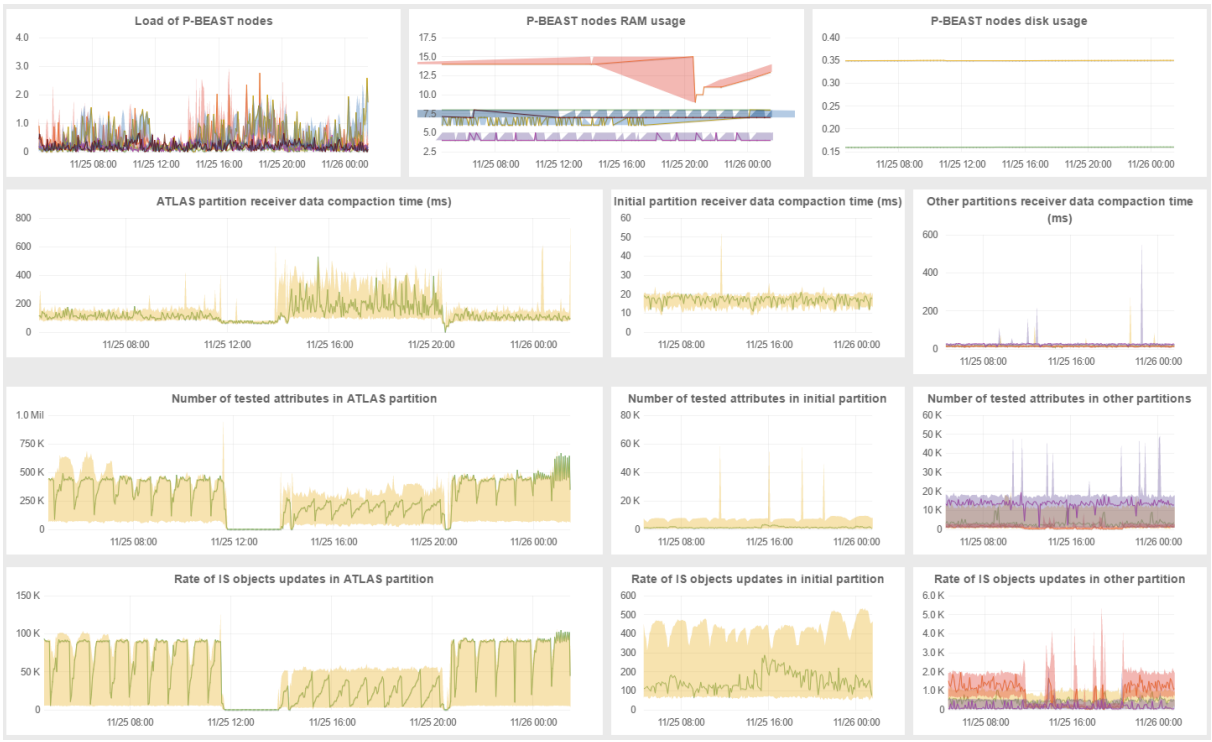

Figure 5. Example dashboard reporting the status of the P-BEAST service during data taking.

5. Summary and outlook

Since their first introduction, P-BEAST and the *Grafana* dashboards have proven to be extremely effective tools to assess the status of the ATLAS TDAQ system. Nowadays dashboards are regularly consulted by users and experts both for real-time monitoring and post-mortem analysis.

Nevertheless, several improvements are planned for the future:

- Move from *Grafana* 1.9.1 to *Grafana* version 3, exploiting its new available functionalities, like the introduction of new graph types (e.g. histograms, pie chart), tables and LDAP integration;
- Introduce support in P-BEAST for some more advanced processing providing data aggregation and manipulation (i.e. direct query for maximum and minimum values, sums, percentages);

- Extend the P-BEAST domain towards detector control, network infrastructure and computing farm monitoring.

References

[1] Boutsioukis G, Hauser R, Kolos S 2012 High-performance scalable information service for the ATLAS experiment *J. Phys. Conf. Ser.* **396** 012026

[2] The ATLAS TDAQ Collaboration 2016 The ATLAS data acquisition and high level trigger system *JINST* **11** P06008

[3] The ATLAS Collaboration 2008 The ATLAS experiment at the CERN Large Hadron Collider *JINST* **3** (2008) S08003

[4] CORBA http://www.corba.org/

[5] Kolos S, Lehmann-Miotto G, Magnoni L, Sicoe A, Soloviev I 2012 A persistent back-end for the ATLAS TDAQ online information service (P-BEAST) *J. Phys. Conf. Ser.* **368** 012002

[6] Carpenter J, Hewitt E, 2016 *Cassandra: the definitive guide* (Sebastopol: O'Reilly Media)

[7] Peters AJ, Sindrilaru EA, Adde G EOS as the present and future solution for data storage at CERN J. Phys.: Conf. Ser. **664** (2015) 042042

[8] Protocol Buffers https://developers.google.com/protocol-buffers/

[9] Sicoe A, Soloviev I New persistent back-end for the ATLAS online information service *Proc. 19th IEEE-NPSS Real Time Conf.,* May 2014, Nara, Japan

[10] CERN Swan Service https://swan.web.cern.ch/

[11] Santos A, Anders G, Avolio G, Kazarov A, Lehmann Miotto G, Soloviev I A validation system for the Complex Event Processing directives of the ATLAS Shifter Assistant J.Phys.Conf.Ser. **664** (2015) no.6, 062055

[12] Grafana http://grafana.org/

[13] AngularJS https://angularjs.org/

[14] Intel i7-3770 http://ark.intel.com/products/65719/Intel-Core-i7-3770-Processor-8M-Cache-up-to-3_90-GHz

[15] CERN Scientific Linux 6 http://linux.web.cern.ch/linux/scientific6/