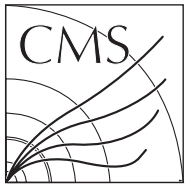
Available on CMS information server

CMS CR -2018/272



The Compact Muon Solenoid Experiment

Conference Report

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland

12 October 2018 (v4, 09 January 2020)

A scalable online monitoring system based on Elasticsearch for distributed data acquisition in CMS

Dainius Simelevicius for the CMS Collaboration

Abstract

The part of the CMS data acquisition (DAQ) system responsible for data readout and event building is a complex network of interdependent distributed applications. To ensure successful data taking, these programs have to be constantly monitored in order to facilitate the timeliness of necessary corrections in case of any deviation from specified behaviour. A large number of diverse monitoring data samples are periodically collected from multiple sources across the network. Monitoring data are kept in memory for online operations and optionally stored on disk for post-mortem analysis. We present a generic, reusable solution based on an open source NoSQL database, Elasticsearch, which is fully compatible and non-intrusive with respect to the existing system. The motivation is to benefit from an off-the-shelf software to facilitate the development, maintenance and support efforts. Elasticsearch provides failover and data redundancy capabilities as well as a programming language independent JSON-over-HTTP interface. The possibility of horizontal scaling matches the requirements of a DAQ monitoring system. The data load from all sources is balanced by redistribution over an Elasticsearch cluster that can be hosted on a computer cloud. In order to achieve the necessary robustness and to validate the scalability of the approach the above monitoring solution currently runs in parallel with an existing in-house developed DAQ monitoring system.

Presented at *CHEP 2018 Computing in High-Energy Physics 2018*

A Scalable Online Monitoring System Based on Elasticsearch for Distributed Data Acquisition in CMS

Jean-Marc Andre[5], *Ulf* Behrens[1], *James* Branson[4], *Philipp* Brummer[2,10], *Olivier* Chaze[2], *Sergio* Cittolin[4], *Diego Da Silva* Gomes[2], *Georgiana-Lavinia* Darlea[6], *Christian* Deldicque[2], *Zeynep* Demiragli[6], *Marc* Dobson[2], *Nicolas* Doualot[5], *Samim* Erhan[3], *Jonathan Richard* Fulcher[2], *Dominique* Gigi[2], *Maciej* Gladki[2], *Frank* Glege[2], *Guillelmo* Gomez-Ceballos[6], *Jeroen* Hegeman[2], *Andre* Holzner[4], *Mindaugas* Janulis[2,9], *Michael* Lettrich[2], *Audrius* Mecionis[5,9], *Frans* Meijers[2], *Emilio* Meschi[2], *Remigius K.* Mommsen[5], *Srecko* Morovic[5], *Vivian* O'Dell[5], *Luciano* Orsini[2], *Ioannis* Papakrivopoulos[7], *Christoph* Paus[6], *Petia* Petrova[2], *Andrea* Petrucci[8], *Marco* Pieri[4], *Dinyar* Rabady[2], *Attila* Racz[2], *Valdas* Rapsevicius[5,9], *Thomas* Reis[2], *Hannes* Sakulin[2], *Christoph* Schwick[2], *Dainius* Simelevicius[2,9,*], *Mantas* Stankevicius[5,9], *Cristina* Vazquez Velez[2], *Michail* Vougioukas[2], *Christian* Wernet[2], and *Petr* Zejdl[2,5]

[1]DESY, Hamburg, Germany
[2]CERN, Geneva, Switzerland
[3]University of California, Los Angeles, Los Angeles, California, USA
[4]University of California, San Diego, San Diego, California, USA
[5]FNAL, Batavia, Illinois, USA
[6]Massachusetts Institute of Technology, Cambridge, Massachusetts, USA
[7]Technical University of Athens, Athens, Greece
[8]Rice University, Houston, Texas, USA
[9]Vilnius University, Vilnius, Lithuania
[10]Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract. The part of the CMS Data Acquisition (DAQ) system responsible for data readout and event building is a complex network of interdependent distributed applications. To ensure successful data taking, these programs have to be constantly monitored in order to facilitate the timeliness of necessary corrections in case of any deviation from specified behaviour. A large number of diverse monitoring data samples are periodically collected from multiple sources across the network. Monitoring data are kept in memory for online operations and optionally stored on disk for post-mortem analysis. We present a generic, reusable solution based on an open source NoSQL database, Elasticsearch, which is fully compatible and non-intrusive with respect to the existing system. The motivation is to benefit from an off-the-shelf software to facilitate the development, maintenance and support efforts. Elasticsearch provides failover and data redundancy capabilities as well as a programming language independent JSON-over-HTTP interface. The possibility of horizontal scaling matches the requirements of a DAQ monitoring system. The data load from all sources is balanced by

* Corresponding author: dainius.simelevicius@cern.ch

redistribution over an Elasticsearch cluster that can be hosted on a computer cloud. In order to achieve the necessary robustness and to validate the scalability of the approach the above monitoring solution currently runs in parallel with an existing in-house developed DAQ monitoring system.

1 Introduction

The Compact Muon Solenoid (CMS) [1] is a general-purpose particle detector at the Large Hadron Collider (LHC) [2] at CERN in Geneva, Switzerland. The CMS Data Acquisition (DAQ) system [3] is responsible for building and filtering of events from about 600 data sources at a maximum trigger rate of 100 kHz. The monitoring system has to provide the facilities to retrieve, process, store, and display monitoring data during the operation of the CMS experiment. The DAQ is composed of a few hundred hosts [4] and of $O(1000)$ interdependent applications that need to be monitored in near real-time. In addition to all requirements, scalability is a major factor that affects all aspects of the system design.

The main purpose of the CMS monitoring system is to collect all necessary data values from all distributed DAQ applications and make them available for further use to diagnose deviation from nominal system behaviour. The data are made available for both online and post-mortem analysis. The approach proposed in this paper fits these needs by providing a scalable solution based on Elasticsearch technology [5]. Another Elasticsearch based system is used for the High-Level-Trigger (HLT) part of the DAQ system monitoring, which also benefits from this off-the-shelf solution facilitating data storing and load balancing [6].

2 Architecture and Design

The CMS DAQ system is a large distributed system based on XDAQ middleware [7-9]. In this context multiple intercommunicating processes in a computer network are running diverse applications, like sub-detector electronics controllers, frontend readout and event builder to accomplish the DAQ task.

The current monitoring system is based on CMS in-house protocols and components following a scalable publisher/subscriber with message broker topology [10-11]. In this approach each process, besides DAQ functionality, includes a dedicated monitoring application, namely, *sensor* that retrieves all the required monitoring information and publishes to a distributed message broker. Collector and data access applications subscribe to the message broker to receive information for collection and, optionally, storage in Oracle database. Retrieved data are then made available to monitoring clients for presentation and analysis to end-users. All the monitoring data are treated using a uniform table-based data format throughout the whole processing chain. Table definitions enumerating all data items from DAQ applications are called *flashlists* and are specified in XML files. Specifications of flashlists reliably identify the content with additional information including timestamp, version and identification fields (see Figure 1). Flashlist format is well suited to define both simple and complex data types, e.g. structures and arrays. The sensor can process different flashlist types and send retrieved monitoring data tables in binary format to the message broker.

2.1 The Elasticsearch Approach

The new approach proposed in this paper benefits from the advantages of using an off-the-shelf technology to define, report, collect, store and conveniently access monitoring data and is a potential replacement to the publisher/subscriber with message broker solution described

above. In this respect Elasticsearch offers the necessary capabilities and functionalities. Definition of monitoring data is implemented with index mapping types. Reporting and collection of data correspond to insertion of documents into indices, whereas data access is achieved with efficient search operation. A standard uniform protocol based on JSON-over-HTTP is used for all Elasticsearch operations, which is suitable for CMS DAQ monitoring system. In addition, Elasticsearch provides required scalability, fault tolerance and data redundancy capabilities to ensure higher reliability.

```
<?xml version='1.0'?>
<xmas:flash xmlns:xmas="http://xdaq.web.cern.ch/xdaq/xsd/2006/xmas-10" id="urn:xdaq-
flashlist:ferolInputStream" version="1.0" key ="context,lid,slotNumber,streamNumber">
        <xmas:item name="context" … type="string"/>
        <xmas:item name="timestamp  … type="time"/>
        <xmas:item name="lid" … type="string"/>
        <xmas:item name="slotNumber" … type="unsigned int 32"/>
        <xmas:item name="streamNumber" … type="unsigned int 32"/>
        …
        <xmas:item name="EventCounter" … type="unsigned int 64"/>
        <xmas:item name="TriggerNumber" … type="unsigned int 32"/>
        <xmas:item name="BX" … type="unsigned int 32"/>
        <xmas:item name="SLinkCRCError" … type="unsigned int 64"/>
</xmas:flash>
```

Figure 1. An excerpt from an example flashlist defining different data fields from CMS ferolInputStream XDAQ application.

The integration of Elasticsearch in the existing framework is implemented by a pluggable application called *timestream* that runs in parallel with the existing sensor application. It retrieves the required local monitoring variables and injects the data into an Elasticsearch cluster using document index operations. In this approach we are able to replace existing in-house components like message broker, collector, data access and store services with Elasticsearch (see Figure 2).

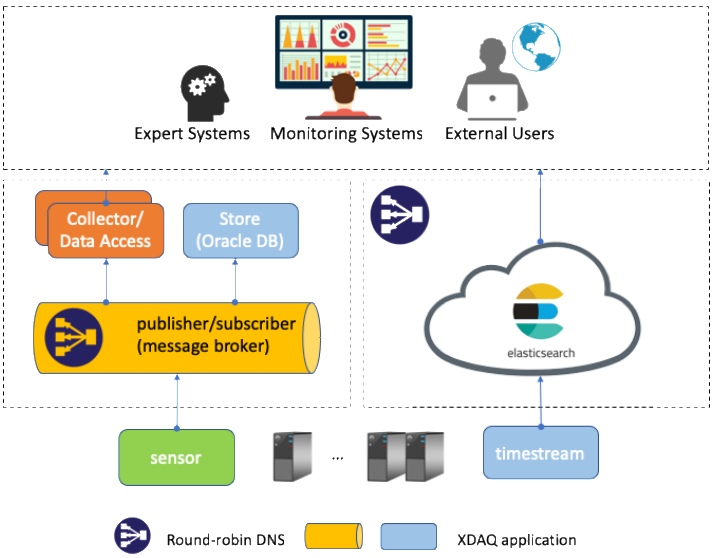


Figure 2. Monitoring data are injected into Elasticsearch cluster from multiple data sources. The data are retrieved directly from Elasticsearch cluster by multiple monitoring clients.

Three types of operations are performed within the monitoring system according to three different roles: data producer, data consumer and data administrator. Operations are executed independently without the need of a central point of coordination. Data producer role is

played by the timestream application, which creates settings and data. Data consumer can be any monitoring client that performs data retrieval. Data administrator is a script that runs periodically within each node of the Elasticsearch cluster and performs maintenance of indices.

The timestream application converts monitoring data variables within the framework into JSON formatted documents that are subsequently injected into Elasticsearch cluster using HTTP protocol. The process consists of three steps: (1) creation of an index template with a mapping for every flashlist, (2) creation of an associated index per predefined time interval, and (3) an actual document insertion into the current associated index. Time-based indices are used in order to allow a document removal by means of a removal of a whole index as opposed to a removal of single documents.

Templates are created only once for each flashlist. They contain information about data mapping as well as general index properties. This way each index is subsequently created based on the information provided in the template.

The timestream automatically creates a new index upon start of a new interval of time. This process results in a sequence of independent indices along the time. The total available data span over all created indices (see Figure 3). Time intervals can be individually configured for each flashlist. The amount of data inserted into each index depends on the data insertion rate and the number of sources within the system.

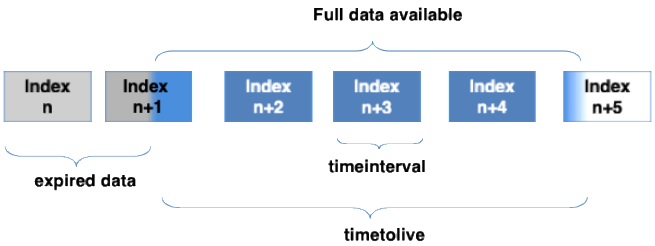


Figure 3. Multiple time-based indices are created dynamically by the timestream application according to a predefined *time interval*.

For each input document an index name is generated according to the following naming convention:

<zone>-<type>_<start time>_<time interval>

where

- *start time* is the beginning of the interval calculated according to the following formula: *start time = current time - (current time % time interval)*;
- *time interval* is a duration of an index as defined by a configuration (both *start time* and *time interval* values are based on ISO 8601 format [12]);
- *type* is a flashlist name;
- *zone* allows a multi-tenant approach to cope with monitoring sources that belong to different systems. This is an example of an index name:

cdaq-ferolinputstream_2018-10-08t01:05:00z_p1dt7h5m

Upon creation of an index data can be inserted. In addition to user data, each document must include the following fields containing timing information: *creation time*, *expiration time* and *withdraw time* as defined by the following expressions:

$$creation\ time = current\ time$$
$$expiration\ time = current\ time + time\ to\ live$$
$$withdraw\ time = current\ time + flash\ interval$$

where

- *creation time* is the time when a document was created;

- *expiration time* is the time when a document expires;
- *time to live* is the document lifespan;
- *withdraw time* is the time when a document becomes too old for instant monitoring;
- *flash interval* is the interval within which a document is searchable for instant monitoring.

The *time interval*, *time to live* and *flash interval* are defined within the configuration file. If the data is not renewed by the data source within the flash interval it means that instant monitoring data is missing.

Data consumers retrieve monitoring data by performing search operations provided by the Elasticsearch API. Since the data are distributed over several indices the search can be performed by using aliases associating a set of required indices. Two aliases are used to refer to different groups of indices (see Figure 4).

One alias refers to all created indices for a particular flashlist. The second one refers to the last two created indices and is used to access the very latest data for efficient instant monitoring. Aliases are automatically created through the template mechanism provided by Elasticsearch upon index creation.

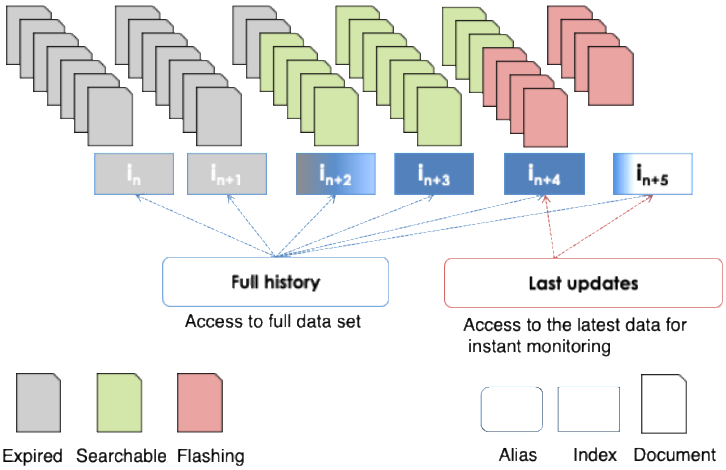


Figure 4. Usage of time-based indices and aliases.

Instant monitoring is used in CMS to retrieve the last monitoring updates from a set of independent DAQ applications. For example, all the counters from all readout units participating in the event building. For this purpose, the *last updates* aliases can be used to limit the search to a smaller number of indices. Further data reduction from the result of the search operation can be obtained by applying a filter according to the *withdraw time*. For search operations over all indices the *full history* alias is used with filtering according to the *expiration time*.

Indices and aliases for a given flashlist need to be maintained according to the configuration. Aliases of instant monitoring pointing to obsolete indices need to be deleted and indices containing expired documents have to be removed. For this purpose, a separate facility called *data administrator*, running on each node of the Elasticsearch cluster is used.

Documents are permanently deleted according to their expiration time. However, in Elasticsearch technology, data removal by means of deletion of a full index is more efficient as compared to the removal of individual documents [5]. Therefore, deletion of expired documents is always achieved by the deletion of an index as a whole with negligible impact on the performance. As a consequence, expired documents are not deleted immediately but

only when every document in an index is expired. Typical time to live spans from 3 days up to 2 years. The deletion of an index is executed according to the following rule:

start time + time interval + time to live < current time

To support efficient data retrieval the aliases pointing to latest indices in the sequence must be maintained. The data administrator script is performing this task by periodically removing aliases to all but the last two indices in the creation sequence (see Figure 4).

Every actor in the system: timestream and data administrator work independently within the system. The consistency of the data is ensured by the following general constraints:

1. The flash interval must be configured long enough to contain the data from all the data sources.
2. Flash interval should be shorter or equal to time to live to prevent the loss of current data entries (*flash interval <= time to live*).
3. The most recent entries needed for instant monitoring must always fit in the last two indices created (*flash interval <= time interval*).

2.2 Performance Management

In order to achieve the required performance, four different issues need to be addressed: (1) the distribution of the input load from all data sources; (2) the expensive Elasticsearch operation of index creation; (3) the specific Elasticsearch internal settings and tuning; (4) the usage of Elasticsearch bulk operations whenever possible.

A large number of sources need to inject monitoring data into several Elasticsearch nodes in parallel. In order to achieve the right balance, every node should have an equal number of network connections. This is achieved by using a DNS round robin scheme where every timestream application that wants to connect to the Elasticsearch cluster is dynamically assigned an IP address of one of the nodes in the cluster. The same approach is used by all monitoring clients that need to retrieve data from the Elasticsearch cluster.

The timestream repeatedly creates indices according to the timing configuration for each flashlist. When creating indices, the status of the Elasticsearch health (green, yellow or red) is assessed [5], before injecting documents. A backoff [13] algorithm with configurable timeout is applied by the timestream application to prevent exceptions when indexing data. During a backoff time, which is of the order of a few seconds, timestream uses internal buffer to temporarily absorb data from data sources.

Additional tuning is necessary to obtain the desired performance. The operating system swapping needs to be disabled in order to avoid a swapping of the Elasticsearch process. The Elasticsearch transaction log durability setting for each index should be set to *async* mode, which instructs Elasticearch to buffer operations in memory and fsync on the regular intervals, rather than fsyncing on each operation. By default, mapping parameters *index*, *store* and *doc_values* of user data fields specified in the flashlist are disabled as they introduce some index-time overhead. These settings can be overridden by the user by specifying the required value in the flashlist XML file. *Index* option controls whether field values are searchable. *Store* option needs to be enabled in cases when the value of a single field or a few fields need to be retrieved instead of the whole document. *Doc_values* is needed if field values are used for sorting or aggregations [5].

Elasticsearch bulk operations increase the efficiency and performance [5]. However, data bulking is not always possible due to the fact that monitoring in CMS must be delivered to monitoring clients as soon as it becomes available, that is, the latest data with the least latency possible. Therefore, the timestream application benefits from the bulk option to improve indexing efficiency only when multiple input sources of the same type are available within the same process.

3 Benchmarking

Benchmarking tests were executed to measure the performance of the system in a systematic way. A test application called *loadtest* was developed as an emulator of a user application. Loadtest is able to inject data at a predefined rate and bulk sizes. Flashlists of four different sizes and of two different data types: *long* and *string* were utilised for the measurements. The benchmarking measurements determined the maximum rate accepted by the system for every combination of data sizes and types (see Figure 5 and Figure 6).

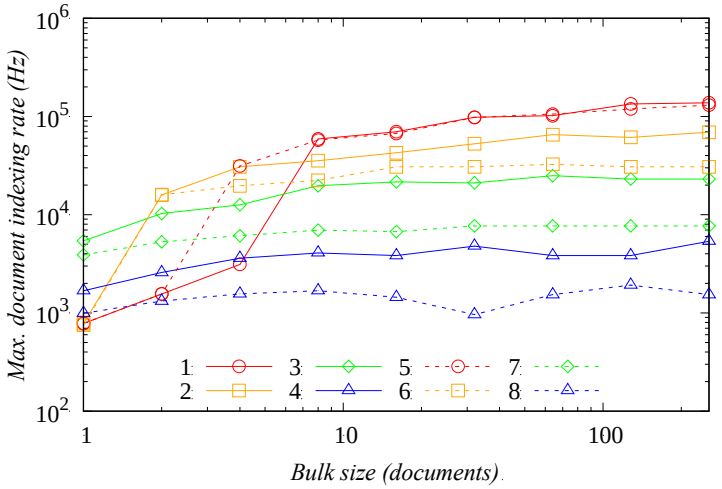


Figure 5. Dependence of the maximum aggregate document indexing rate on the bulk size for the flashlists with a different number of user defined fields (1 field (1, 5), 16 fields (2, 6), 64 fields (3, 7), 256 fields (4, 8)) of type *long* (1, 2, 3, 4) and *string* (5, 6, 7, 8).

Tests were performed using a cluster of Elasticsearch which consisted of 5 nodes. Data were injected from 30 data sources, each running on a separate physical machine with separate processes for timestream and loadtest applications. Before each measurement all processes running timestream and loadtest applications were stopped and all indices and templates were deleted from Elasticsearch. The document indexing rate was calculated by

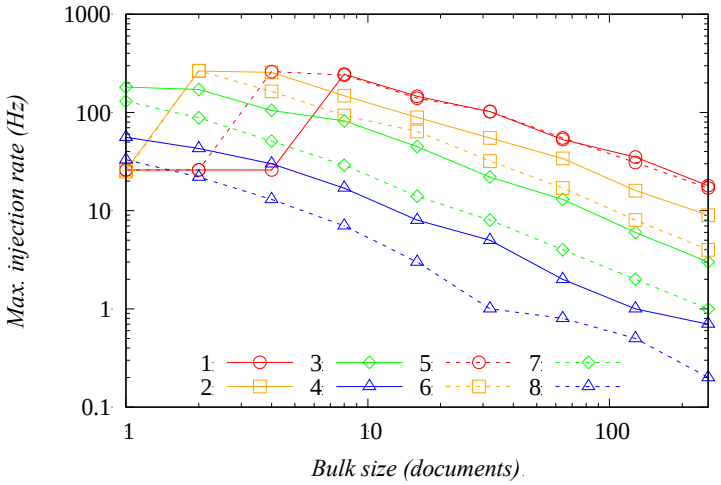


Figure 6. Dependence of the maximum injection rate from a single data source on the bulk size for the flashlists with a different number of user defined fields (1 field (1, 5), 16 fields (2, 6), 64 fields (3, 7), 256 fields (4, 8)) of type *long* (1, 2, 3, 4) and *string* (5, 6, 7, 8).

reading two *index_total* values from statistics provided by Elasticsearch API. The results of maximum aggregate document indexing rate achieved at different bulk sizes are depicted in Figure 5. The same data were used to plot the dependence of maximum injection rate from a single data source on the bulk size (see Figure 6).

Figure 5 shows that the highest document indexing rate can be achieved when using bulking feature of Elasticsearch API with moderate and high bulk sizes. Smaller documents can be indexed faster. For small documents non-systematic behaviour was observed at the low bulk sizes and in the absence of bulking. The maximum document indexing rate was measured equal for the smallest flashlists with fields of both long and string type. There was a difference in maximum rate when the measurement was performed with bigger flashlists. Maximum rate was smaller for flashlists with strings. However, from this test it is not clear if this difference is caused by the difference in efficiency of indexing different types of data or is it caused by the size of the document. Each value was a *long* of 8 bytes or a 32-character *string* randomly generated for each injection. As shown in Figure 6 for the biggest documents (with 256 user defined fields) and at the highest bulk size, the injection rate on individual data sources had to be reduced below 1 Hz.

4 Summary

Elasticsearch was shown to be a valuable technology to support CMS DAQ monitoring. The proposed solution is used in parallel with the existing monitoring system with comparable performance and functionality. The measured document indexing rate for the full CMS DAQ monitoring system generated by 265 sources in parallel is about 2 kHz with an average document size of 26 fields of different types (813 fields in total and 31 flashlists), bulk size varies from 1 to 180 throughout the system. This rate is about one order of magnitude lower than the measured maximum performance and is well accepted by the Elasticsearch cluster with no data losses with reading clients running in parallel as required by CMS DAQ monitoring. Data persistency is implicitly provided by defining an arbitrary value of time to live. The solution demonstrated the advantage of using a uniform protocol for reading and writing data, which facilitated the integration of Elasticsearch into the existing system. The use of Elasticsearch moves the problem of in-house setup and expertise of the monitoring system to a mere system administration exercise.

References

1. The CMS Collaboration, *The Compact Muon Solenoid: Technical Proposal* (1994)
2. The LHC Study Group, *The Large Hadron Collider: Conceptual Design* (1995)
3. The CMS Collaboration, *The Trigger and Data Acquisition Project* (2002)
4. T. Bawej, et al., IEEE Trans. Nucl. Sc., **62**(3), 1099-1103 (2015)
5. Elasticsearch, https://www.elastic.co
6. J.-M. Andre, et al., J. Phys. Conf. Ser., **664**(8), 082036 (2015)
7. V. Brigljevic, et al., Using XDAQ in Application Scenarios of the CMS experiment, *Conference for Computing in High-Energy and Nuclear Physics - CHEP03* (Stanford Linear Accelerator Center, 2003)
8. J. Gutleber, S. Murray, L. Orsini, Comp. Phys. Comm., **153**(2), 155-163 (2003)
9. J. Gutleber, L. Orsini, Clust. Comp. **5**(1), 55-64 (2002)
10. G. Bauer, et al., J. Phys. Conf. Ser., **219**(2), 022042 (2010)
11. D. A. Chappell, *Enterprise Service Bus* (O'Reilly, 2004)
12. ISO 8601:2004(E), Data elements and interchange formats — Information interchange — Representation of dates and times (2004)
13. IEEE 802.3-2015 - IEEE Standard for Ethernet (2015)