A scalable monitoring for the CMS Filter Farm based on elasticsearch

2015 J. Phys.: Conf. Ser. 664 082036

(http://iopscience.iop.org/1742-6596/664/8/082036)

Journal of Physics: Conference Series **664** (2015) 082036 doi:10.1088/1742-6596/664/8/082036

A scalable monitoring for the CMS Filter Farm based on elasticsearch

J-M Andre[5], A Andronidis[2], U Behrens[1], J Branson[4], O Chaze[2], S Cittolin[4], G-L Darlea[6], C Deldicque[2], M Dobson[2], A Dupont[2], S Erhan[3], D Gigi[2], F Glege[2], G Gomez-Ceballos[6], J Hegeman[2], A Holzner[4], R Jimenez-Estupiñán[2], L Masetti[2], F Meijers[2], E Meschi[2], R K Mommsen[5], S Morovic[2], C Nunez-Barranco-Fernandez[2], V O'Dell[5], L Orsini[2], C Paus[6], A Petrucci[2], M Pieri[4], A Racz[2], P Roberts[2], H Sakulin[2], C Schwick[2], B Stieger[2], K Sumorok[6], J Veverka[6], S Zaza[2] and P Zejdl[2]

[1] DESY, Hamburg, Germany
[2] CERN, Geneva, Switzerland
[3] University of California, Los Angeles, California, USA
[4] University of California, San Diego, California, USA
[5] FNAL, Chicago, Illinois, USA
[6] Massachusetts Institute of Technology, Cambridge, Massachusetts, USA

E-mail: srecko.morovic@cern.ch

Abstract. A flexible monitoring system has been designed for the CMS File-based Filter Farm making use of modern data mining and analytics components. All the metadata and monitoring information concerning data flow and execution of the HLT are generated locally in the form of small documents using the JSON encoding. These documents are indexed into a hierarchy of elasticsearch (es) clusters along with process and system log information. Elasticsearch is a search server based on Apache Lucene. It provides a distributed, multitenant-capable search and aggregation engine. Since es is schema-free, any new information can be added seamlessly and the unstructured information can be queried in non-predetermined ways. The leaf es clusters consist of the very same nodes that form the Filter Farm thus providing natural horizontal scaling. A separate central" es cluster is used to collect and index aggregated information. The fine-grained information, all the way to individual processes, remains available in the leaf clusters. The central es cluster provides quasi-real-time high-level monitoring information to any kind of client. Historical data can be retrieved to analyse past problems or correlate them with external information. We discuss the design and performance of this system in the context of the CMS DAQ commissioning for LHC Run 2.

1. Introduction

The Compact Muon Solenoid (CMS) detector in the run 2 after the Long Shutdown 1 (LS1) features a redesigned Data Acquisition System [1], replacing the Myrinet-based event-builder with the system based on Infiniband and 40 Gbit Ethernet networks[2]. The full event building in DAQ2 is performed on Builder Units (BU) which are forwarding the event data to Filter Unit (FU) nodes running the High Level Trigger software. BU and FU nodes are connected using 40 Gbit Ethernet link on the BU side and either 10 Gbit (new generation of FU nodes) or 1 GBit

(legacy HLT nodes) on the FU side. The builder unit and corresponding FU nodes form a HLT appliance. There are 62 appliances in the DAQ2 system, consisting of between 12 and 18 FU nodes each.

2. Run 2 filter farm

FU nodes are multi-core machines running standard offline CMS Software Framework (CMSSW) jobs. In run 1 a special software wrapper was used to run CMSSW in the same process as the Online CMS Framework (XDAQ). In run 2 there is a clear separation between the two frameworks. XDAQ runs only on BU nodes, while CMSSW runs only on FU nodes. To establish data transfer of events built on the BU, a File-based approach is used[3]. FU nodes use NFS v4 to mount a ramdisk partition on a BU which serves as a large (240 GB) buffer of fully built events (raw data). FU nodes use a file-locking mechanism to reserve a raw data file which they will be processing. A file naming convention is used to mark that a data file belongs to a particular *lumisection*, an interval of approximately 23 seconds.

Each raw data file in the BU ramdisk is accompanied by a JSON metadata file, describing number of events in a file. HLT processes executing on FU nodes output their data at the end of a lumisection in multiple output *streams*, also providing accounting of processed events and events selected by the HLT in JSON metadata files.

There are three merging steps performed in the Filter Farm and the Storage system[4], where data and metadata are aggregated and checkpointed at the end of each lumisection. It is verified that the processing is completed and that all output of the previous step is present. First such step, called micro-merging, occurs at the FU node where output of HLT processes is merged. Second step, mini-merging, is performed at the BU node and involves merging the output of all FU nodes in the appliance. Finally, a global merging step, the macro-merging, is performed in the global Lustre filesystem where data from all appliances is merged. JSON documents are used to describe input data of each of these steps and their event accounting is compared against content of input JSON files written in ramdisk. A new JSON document summarizing merged data is produced at each merging step and is used as input for the next step.

3. Monitoring of the Filter Farm

JSON files produced by the Filter Farm fully describe the event flow of the Filter Farm and can be used to monitor the traffic and status of the event processing and merging in the system. It is thus desirable for the monitoring system to handle the JSON format natively. The system should also scale well to a full Filter Farm size (approx. 1000 nodes) and provide a quasi-realtime response, because a fast feedback is required for Online activities and troubleshooting of the system. elasticsearch[5] was selected as a suitable technology for this use case.

Elasticserch is a multitenant NoSQL database and a search engine based on the Apache Lucene. It provides a quasi-realtime document insertion and search capabilities. JSON format is supported natively for both document storage and as a building block of the RESTful API, supported over HTTP and several other protocols. Building a distributed system is supported with arbitrary number of nodes forming a cluster. Clusters can be established seamlessly by assigning a cluster name to each of the nodes. Nodes then automatically elect a master node, which is responsible for maintaining the bookkeeping of the cluster state. Documents in elasticsearch are stored in named indices, each being split into multiple shards distributed over the cluster. Indices can be configured to have shard replicas for improved redundancy in case of failure of any of the nodes. Each document is described by a mapping which describes variable types, storage properties (e.g. document expiration time), parent-child relationships between documents, indexing options and other properties. Mapping is commonly inferred automatically from the structure of the inserted documents, however it is also possible to define the mapping explicitly for a more precise control of data types and indexing options.

Searches can be performed by querying any of the cluster nodes, upon which the search request is distributed to each node which contains a shard or replica belonging to the index being searched. A large number of query types is available, for example searching based on certain term values or searching in specific intervals of date/time. A more advanced form of searching can be done by using aggregation based on algorithms, for example to build histogram documents with time-series bins, which is internally built by binning document information based on their timestamps.

3.1. Central cluster

A dedicated elasticsearch cluster consisting of approx. 20 nodes is used to run a central elasticsearch service (central-ES), which is the main store of the Filterfarm monitoring information. The cluster allocates more than 10 GB of the working memory per node and a suitable disk space to allow permanent storage of the monitoring information which describes the Filter Farm data flow in global runs. FilterFarm status and logs are also stored in dedicated indices. A separate set of indices are used for systems other than the central DAQ, such as MiniDAQ and the validation system. Documents are written using index aliases, which can seamlessly be redirected to a new index revision when necessary, usually when a change in document mapping is required.

All metadata files created on BU nodes and by the macro-merging service are injected directly into the central services by dedicated scripts. Injection is done by inserting JSON documents over the elasticsearch HTTP interface. This includes documents written by the event builder application on BU, which provide the number of events and files built in a lumisection by each BU, as well as mini-merging documents created by a merger running on a BU. Logs from FFF services running on all FU and BU nodes are collected by a local script and injected into the central cluster. BU services also inject "boxinfo" documents, which describe state of each Filter Farm node, such as ongoing runs, available CPU resources or how resources are currently used, or the usage of disk space.

Due to a large number of FU nodes, insertion and storage of all JSON documents appearing from all HLT processes into the central cluster is not feasible without significantly expanding the cluster. Instead, the Filterfarm hardware is exploited to run a local elasticsearch cluster in each appliance, into which all JSON data from FU nodes (with exception of logs from FFF services) is then inserted. Data from BU nodes remains injected directly into the central server. After the information is inserted locally, appliance clusters are queried to extract the data written by FU nodes in an aggregated form which is then written into the central index. Querying is done by a dedicated per-run plugin which collects the information for the duration of the run. To facilitate this, a set of *Tribe* nodes is used to distribute queries to all nodes in the Filterfarm. This structure is described in more detail in the following two sections.

3.2. Appliance clusters

Elasticsearch instance is installed on every BU and FU node profiting from the Filter Farm structure. Instances are configured to form a cluster containing same nodes as an appliance, as illustrated in Figure 1. BU is a data-less master node, while FU nodes are data nodes storing per-run index shards. All nodes are limited to 1 GB JVM heap and have been measured to not have a high CPU usage during a typical HLT run (utilizing O(10%) of a single CPU core).

When a new run is started, a new index is created in each appliance cluster and is used for injection of FU documents for the duration of the run. The index is not created explicitly, but dynamically when a first document is inserted. Mapping of documents is predetermined by a template which specifies how document properties will be defined once first document of a specific type is indexed. Injection of the documents is done by a script that parses JSON metadata documents produced by each HLT process as well as the micro-merging script output.

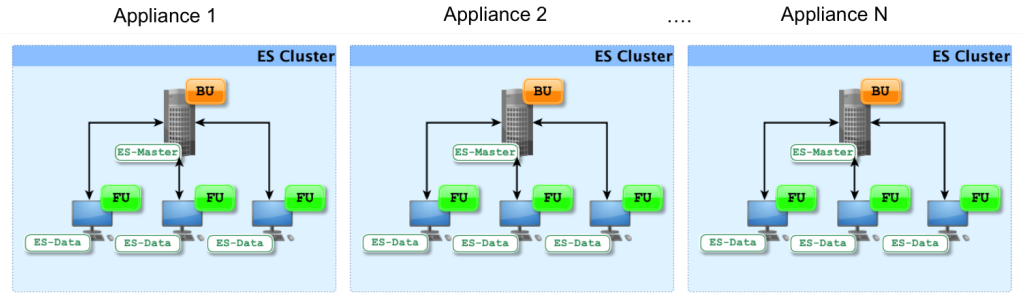

Figure 1. Elasticsearch appliance clusters.

When each input raw data file is taken for processing by a HLT process, the accompanying metadata file (providing number of input file events) appears in the local disk and is subsequently inserted into the local elasticsearch cluster. HLT stream output metadata from each process or from the merged output on FU are also inserted. Each HLT process also runs a monitoring service which samples current HLT state (currently running module) in 1 second intervals and outputs a JSON file that is injected into local cluster. These documents are later used to obtain the CPU usage distribution of the HLT, called the microstate distribution. Logs produced by HLT processes are also injected locally. After the run is finished, the run index can be closed to reduce the amount of resident memory used by the elasticsearch process.

3.3. Tribe of clusters and data collection

In order to query the full Filter Farm, it is necessary to perform a search in each of the appliance clusters. In the system of fully separated clusters, this would require performing an API request to each of the clusters and merge the results by the client-side code. Fortunately, elasticsearch provides a *Tribe* feature which allows a specific node to become a data-less member of multiple clusters simultaneously, without resulting in any mutual visibility between those clusters. This allows API requests sent to the Tribe node to be distributed to the cluster using internal elasticsearch protocols and have query results merged in the same way as for queries that are running on a single cluster. For the Filter Farm monitoring, we use a set of 10 mutually independent Tribe nodes. They share a common DNS alias, with a round-robin rotation used to load balance requests. Each Tribe node is configured to join each of the 62 DAQ2 appliances. [1]

As the information stored in appliances is considered temporary, we aggregate it into a final set of documents in the central index. This is done using aggregation queries which return results in form of the more compact documents that get stored in the central index. The process-level granularity is not preserved in the process. A dedicated plugin, called the RunRiver plugin is used for this task. The plugin is built using the elasticsearch River plugin API. Plugin's lifecycle is managed by the central ES cluster which guarantees that the plugin runs in the cluster. In case of a failure of the node running the plugin, the plugin is relocated to another node automatically. A main instance of the plugin runs permanently and is responsible for detection of the new run. Separate main instances are used to monitor other DAQ systems. On each new run, a separate plugin instance responsible for run data aggregation is started, as illustrated in Figure 2. This

[1] Issues were found when using the Tribe feature with many nodes present in the DAQ cluster. A Tribe node maintains connection to each node in the cluster and therefore requires many (mostly inactive) threads. To allow connection to more than 100 nodes, this required the default process limit to be raised for the elasticsearch user, from 1024 to the order of 10000. Another issue was that the Tribe was not gracefully handling closing of indices in appliances. We identified this flaw in the Tribe class implementation, with the fix subsequently merged upstream.

instance runs specific aggregation queries on the whole cluster and stores compact histogram-like documents which are small enough to be used by a real-time monitoring UI.

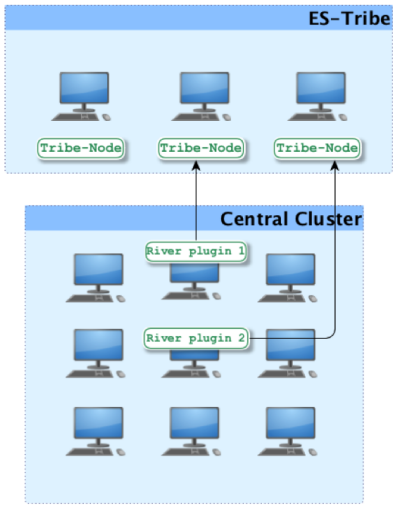

Figure 2. RunRiver plugin instances running in the central server.

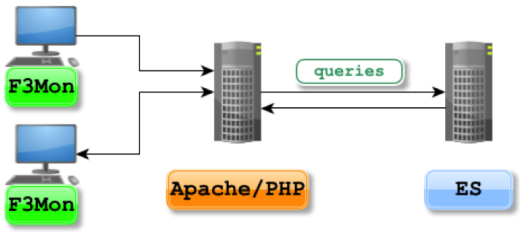

Figure 3. Web monitoring UI infrastructure.

4. Monitoring User Interface

A web-based monitoring interface, called the F^{3}mon UI, was built on top of the elasticsearch based monitoring system. The purpose of the interface is to serve as a general web page used by both experts and regular users, such as shifters, to assess the current state of Filter Farm. The interface is a browser application built using modern JavaScript libraries such as Bootstrap[6], jQuery[7] and Highcharts[8] (used to build graphs).

The interface communicates with the PHP server which serves as the gateway to the central elasticsearch server, as illustrated in the Figure 3. Direct access to elasticsearch from the browser is avoided, as the elasticsearch API also provides means of data manipulation and interference with the cluster function. Instead, PHP scripts only implement specific type of search requests using only search APIs of elasticsearch. This approach also allows caching of requests for multiple clients displaying the same information, which however is not presently implemented. To provide better scalability, each of the central cluster node runs a PHP server instance locally and is accessed using the same DNS alias.

The interface dynamically sets parameters sent to the PHP server scripts based on the content selected by the user. PHP uses those parameters, such as run number, stream selection, or a lumisection range, to construct and run aggregation queries in the elasticsearch cluster. Results

are parsed and returned as JSON documents in the form suitable for the graphical presentation by the GUI.

4.1. F³Mon UI components

The interface displaying an ongoing run is shown in the Figure 4. By default, the interface detects if there is an ongoing run and will display live information. A full run history is also available from a selector in the lower left part of the UI. When a run from the history (or the current run) is selected, it becomes an active run and is displayed in other components of the UI. General information, such as the list of HLT streams, start and stop time of the run, or a total number of lumisection intervals, is shown in the upper left part of the UI.

The Stream Monitor in the upper right side of the UI shows HLT output rate of each stream aggregated over all appliances for each lumisection. It is possible to display the rate in terms of events per second or a total throughput per second. Completeness of micro-merging in the system is overlaid over the graph. In the lower part of the graph, mini and macro merging completeness is shown, with the green color used to mark that the merging stage is complete. In the given example, yellow color marks lumisections where mini-merging was not fully completed.

The interface also provides a "drill-down" inspection which is useful when a problem, such as the incomplete merging, appears somewhere in the system. By clicking on the mini or macro-merging completion box, the view will switch to display the merging completeness per stream, and subsequently per each HLT appliance.

The microstate monitor is provided in the bottom right part of the interface. It displays the CPU usage breakdown per HLT module, aggregated from each HLT process in the system, which is useful in detecting inefficiencies in the HLT event processing. In the given example, the CPU usage is predominantly idle.

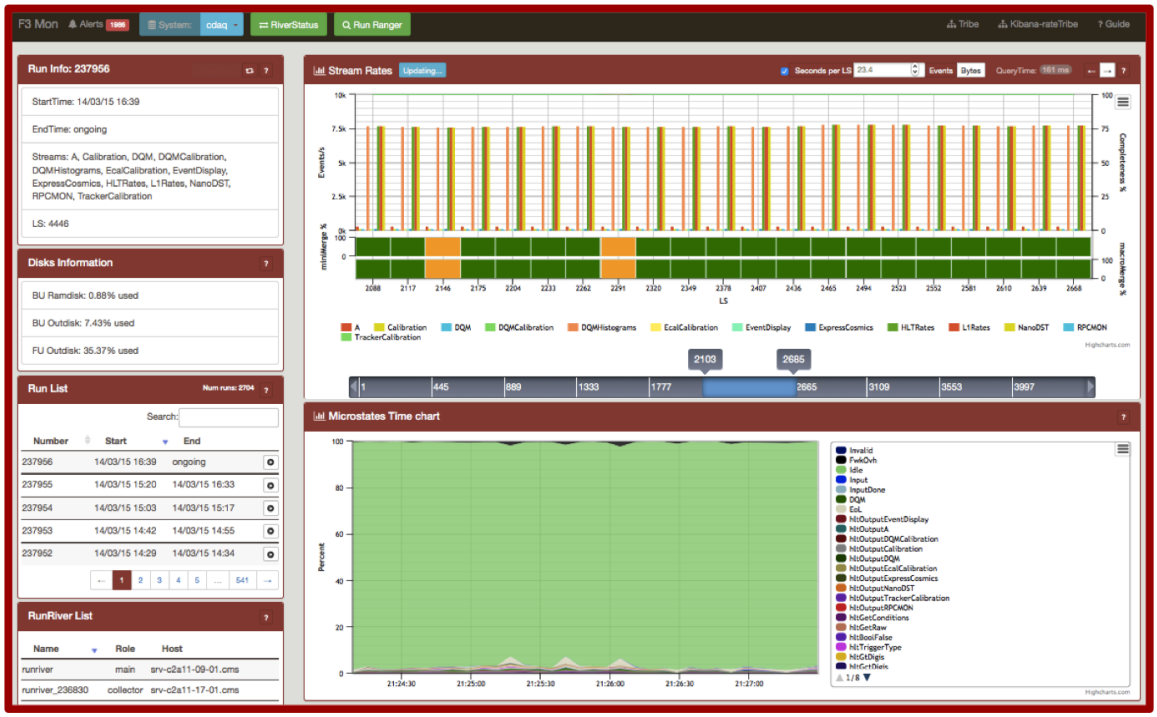

Figure 4. F³Mon Interface. Left side of the interface shows current run information and history of runs selector. Right side shows stream output rates and completition and CPU usage of the HLT for the current run.

4.2. Expert tools

By default, elasticsearch builds index using information from whole document content, so it is possible to cross-correlate data without special provisioning on how documents are stored and indexed. This flexibility has been used to build several tools to diagnose problems in a way that has not been originally intended when injecting data. An example of this is given in Figure 5 showing a web-based tool used to correlate time delays in data merging shown in terms of streams and lumisection intervals. A clear correlation between stream type and file size is seen, however without correlation of those variables with the merging time. The aggregation query used to retrieve a 3-dimensional histogram document is shown in the Figure.

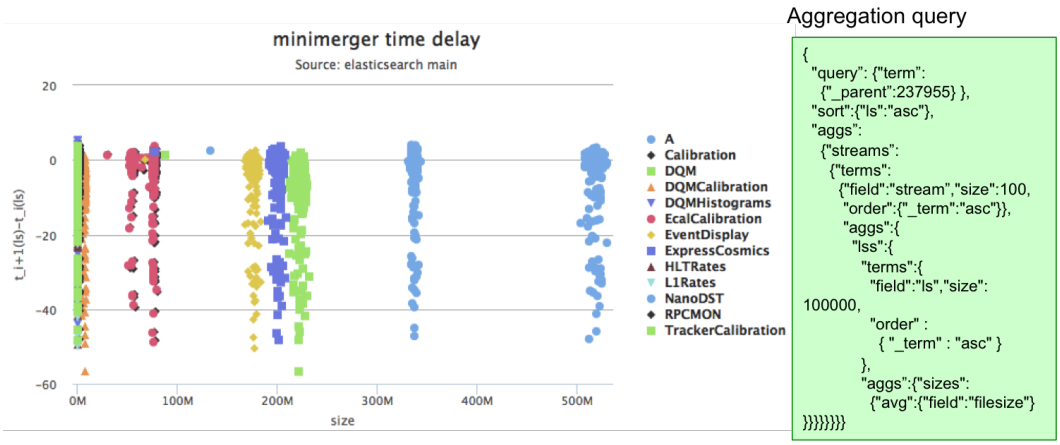

Figure 5. Expert tool showing correlation between the merging latency, stream type and file size. A query used to obtain the shown distribution is shown in the right side.

Service	ElasticSearch Status	Data Nodes / Appliance Nodes	Active Primary Shards	idle slots	active slots	cloud slots	stale FUs no heartbeat in >10s	dead FUs no heartbeat in >1h	disconnected FUs no heartbeat FOUND	ramdisk % used	local (FU) disk % used	output (BU) disk % used
tribe_server	red	634	40853									
appliance_clusters												
bu-c2d31-10-01(243034, 243289) [age=6 s] connected	green	12/12	994	0	192	96	0	0	0	0.17	49.46	0.00
bu-c2d31-20-01(243034, 243289) [age=2 s] connected	green	12/12	994	0	192	96	0	0	0	0.19	49.46	0.00
bu-c2d31-30-01(243034, 243289) [age=2 s] connected	green	12/12	994	0	192	96	0	0	0	0.04	49.46	0.00
bu-c2d32-10-01(243289) [age=1 s] connected	green	12/12	1162	0	192	96	0	0	0	0.08	50.00	0.00
bu-c2d32-20-01(243289) [age=6 s] connected	green	12/12	1162	0	192	96	0	0	0	0.04	50.02	0.00

Figure 6. F^3 general status page. The UI displays the following information: list of the ongoing runs in the appliance, status/problems of elasticsearch cluster, number of present and expected appliance nodes, number of elasticsearch shards in the local cluster, allocation of CPU cores, detection of nodes without the heartbeat, and disk usage statistics of the appliance.

4.3. F^3 general status

Individual cluster nodes and appliances are monitored through the F^3 general status page shown in Figure 6. Similarly to F^{3}mon, it queries elasticsearch through the PHP server. Comparison is done between the relational database information describing the full DAQ system, cluster

information present in elasticsearch, and "boxinfo" documents which describe each HLT node and serve as a heartbeat. Inconsistencies between these information sources are detected and highlighted by the UI. This is highly useful in detecting outage of either the elasticsearch service on a particular node (which can be a source of inconsistent run monitoring), a malfunctioning in non-monitoring Filter Farm software components (e.g. service crash resulting in a missing heartbeat), or a hardware failure (e.g. broken network interface).

5. Summary

CMS has implemented a monitoring system for post-long shutdown 1 that complements the redesigned File-based Filter Farm and takes advantage of elasticsearch, the emerging NoSQL and search engine implementation. By utilizing the indexing of injected information in elasticsearch, we are able to provide a quasi-realtime full-detail insight into event processing information, using same mechanisms to inspect live run information as well as run history.

The Filter Farm monitoring system consists of a dedicated central cluster, serving as a permanent document storage, and clusters following the HLT appliance structure, which store process-level information and reuse Filter Farm hardware to provide scalability with a large number of appliances. Information from appliance clusters is aggregated using Tribe nodes which are members of each cluster and entry points for queries running over the whole cluster. Elasticsearch RunRiver plugin instances running in the central server are responsible for aggregating the information into a more compact form suitable for the monitoring UI.

A general monitoring page, the F3Mon UI, was designed to provide run monitoring status suitable for non-expert use, displaying information such as ongoing run status, completion of the HLT processing and output merging, and CPU usage in the HLT. The interface uses a PHP server running on central ES cluster nodes as as gateway to the central ES cluster.

Several expert tools have been built on top of the same infrastructure. A tool was developed to inspect delays in output merging stages. F3 general status page was developed to help experts investigate status of individual appliances and HLT processing nodes.

In addition, non-web based clients can be served by PHP scripts, such as LabView-based monitoring application which displays HLT CPU usage information.

6. Outlook

The full DAQ2 Filter Farm integration will be completed in time for the LHC run 2 physics program. The system consisting of O(1000) nodes will provide a challenge in reliability of appliance clusters, as well as scalability of Tribe nodes and the central elasticsearch cluster load. While Tribe and central cluster nodes were utilizing retired 8-core HLT nodes used in run 1, those will be replaced by a more recent hardware providing more CPU cores as well as more working memory and disk space.

The CMS DAQ group has also been evaluating elasticsearch as a monitoring storage backend in other areas of DAQ, in particular DAQ readout and event building where the most detailed monitoring information has previously not been saved and was only accessible live.

References

[1] Bauer G et al. The new CMS DAQ system for LHC operation after 2014 (DAQ2) 2014 *J. Phys.: Conf. Ser.* 513 012014
[2] Mommsen R K (these proceedings)
[3] Meschi E (these proceedings)
[4] Darlea L (these proceedings)
[5] elastic https://www.elastic.co
[6] Bootstrap http://getbootstrap.com
[7] jQuery https://jquery.com
[8] Highcharts http://www.highcharts.com