CMS Online Data Quality Monitoring: Real-Time Event Processing Infrastructure



2011 J. Phys.: Conf. Ser. 331 022022

(http://iopscience.iop.org/1742-6596/331/2/022022)

Journal of Physics: Conference Series **331** (2011) 022022 doi:10.1088/1742-6596/331/2/022022

# CMS Online Data Quality Monitoring: Real-Time Event Processing Infrastructure

**Srećko Morović[1] for the CMS DQM Group**

[1]Instute Rudjer Boskovic, Bijenicka cesta 54, HR-10000, Zagreb, Croatia

E-mail: srecko.morovic@irb.hr

**Abstract.** The CMS experiment is one of the large experiments at the LHC at CERN. The CMS online data quality monitoring (DQM) system comprises a number of software components for the distribution, processing and visualization of event data. Already before the year 2010 the system had been successfully developed, deployed and operated in previous data challenges with cosmic ray muons and LHC beams. In preparation for the LHC data taking period of 2010, the performance and efficiency of the infrastructure was evaluated and a number of improvements were implemented with the goal to improve the robustness and data throughput of the system and to minimize the operation and maintenance effort. In this report the main considerations and achieved improvements are described.

## 1. Introduction

The CMS Online Data Quality and Monitoring System (Online DQM)[1] provides live data quality and integrity information based on the processing of event data in real-time, i.e. while the data taking is on-going. The main purpose of the system is the fast and reliable detection of possible hardware and software errors.

Online DQM operates on detector event data, aggregated from the detector's Data Acquisition System (DAQ) [2] [3], and produces histograms of data distributions. In addition to event data, a selected set of conditions data is included for navigation and bookkeeping purposes. The histogram information is delivered to and visualized in real-time by the DQM webserver GUI [4], which is accessible experiment-wide. Regular online DQM shifts are organized for the visual inspection of the DQM data, synchronously during the data taking. The tracking of the regular data quality inspection workflows and the bookkeeping of the produced run quality information is facilitated by the Run Registry [5].

The Offline DQM system is based on histogram output from offline data reconstruction processing and provides final data quality assessment for each reconstruction cycle. Online DQM and Offline DQM are separate systems and use independent instances of the same DQM webserver GUI, and share most of the subsystem specific histogramming code.

## 2. The CMS Data Acquisition System

The recording of events in CMS is triggered by the CMS Level 1 hardware trigger system, based on decisions made from detector data primitives, and/or technical and beam conditions. The Data Acquisition System (DAQ) retrieves the detector raw data from the detector frontend hardware, aggregates them to event data and delivers the event data to the High Level Trigger

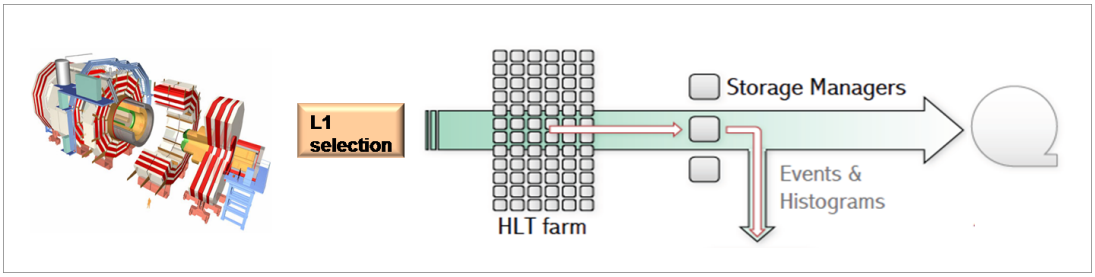


**Figure 1.** Overview of the CMS DAQ system

(HLT). The HLT system is comprised of a large PC cluster and implements a software filter in which further trigger requirements are imposed. Events that are accepted by the HLT system are passed to several instances of Storage Manager (SM) applications, that are dedicated to the storage of the data on disk. For monitoring purposes the SM implement so-called event servers which deliver subsets of the events to client applications on request.

The HLT consists of more than 5000 nodes and is capable to process an input event rate of 100 kHz. The HLT output contains raw event data and additional HLT products, including a summary of trigger results. About 200 events per second are selected and passed to the SM for permanent recording. The HLT also produces a small set of histograms which are passed to and summed within the SM. The online DQM system retrieves events and histograms from the Storage Manager event and histogram servers. Event delivery rates in excess of 100 Hz have been achieved. The data flow in the DAQ system is illustrated in figure 1.

The same data processing software environment (CMSSW) is used for online trigger filtering (HLT), monitoring (DQM), offline event reconstruction and physics data analysis. The CMSSW event processing code is written in the C++ language, comprising of modules that are instantiated and sequentially executed according to configurations provided in Python code. Internally, CMSSW uses the ROOT [6] framework and ROOT is also used to serialize (pack) the event data for data transfer and storage.

All online system applications are implemented as finite state machines, controlled together with the detector hardware from the Run Control and Monitoring System (RCMS)[7]. For the execution of CMSSW code in the online environment, i.e. in the HLT, as well as in online DQM, a special finite state machine application, the Filter Unit Event Processor (FUEP), is used, which interfaces the event processing state transitions within CMSSW with RCMS commands.

## 3. The Online DQM System Architecture

The DQM online system has been designed to provide efficient and fast detector and trigger monitoring based on event data. The main design choices were driven by the requirements of maximum flexibility – e.g. ad-hoc updates of histogramming code – and minimal interference with the data taking, triggering and permanent data storage. Following these requirements, the online DQM system was implemented as a high data throughput spy-mode system for event data processing outside of the main event data filtering and recording stream. In this architecture possible technical failures or performance bottlenecks within DQM are isolated from the upstream DAQ and HLT systems.

The DQM system consists of three major components, sketched in figure 2: the Storage Manager Proxy Servers (SMPS) funnel event data from the several SM to the DQM event consumer applications, the DQM applications create the histograms and related data quality information; and the DQM GUI provides centralized visualization of the DQM histograms.

The DQM GUI receives updates of histogram data from the DQM applications through a

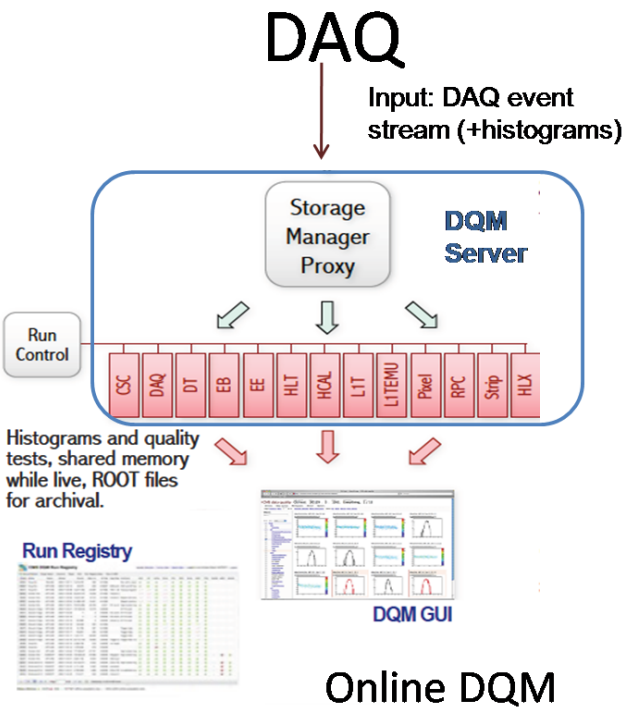


**Figure 2.** Sketch of the online DQM production system as described in section 3.

DQMCollector application [4]. During the data taking, events are continuously processed, and histogram data are continuously updated and shipped to the DQMCollector. After each run the histogram data are archived and indexed for later reference.

Each DQM application is configured individually, using the modular execution sequences within CMSSW, for different subdetector-specific monitoring tasks. Each application is executed on a single core of the multi-core CPU, using the same FUEP applications of the HLT, as described in section 2. The FUEP are configured specifically for event data retrieval and histogram export within DQM, while the FUEP interfaces to the DAQ components upstream of the HLT are disabled. In 2010 the system contained about 25 different DQM consumer applications distributed over five 8-core server machines. In total, the online DQM applications produce as many as 300,000 histograms of which around 50,000 are accessible in real-time in the DQM GUI.

The SMPS decouples the connection to the SM event servers from individual subsystem applications, and maintains a buffer of received events, serving them to clients on HTTP protocol requests. In general, a single SMPS instance is running on each multi-core local machine of the DQM cluster in order to minimize the data transfer over network, and in order to share SM connections for multiple clients. One additional functionality that the SMPS provides is the creation of specific queues in the SM that allow for the selection of very rare trigger paths. This allows for the consumers, interested in those particular trigger paths, to get good sampling, instead of getting their queues overfilled with undesired events.

The online DQM system is integrated into the central run control system (RCMS) through a so-called DQM function manager which is responsible for instantiation and state control of DQM Server applications. The specific DQM system configuration, i.e. the placing of individual C++ event serving and processing applications with particular configurations on the various machines on the DQM PC cluster, and the connectivity of the system with the Storage Managers that

serve the events, is defined through a specific run control configuration.

The whole system comprises three identical systems for production, integration and development, each consisting of a DQM GUI, the full set of DQM applications and corresponding Storage Manager Proxy Servers. The production system is integrated in the run control cycles of the CMS data taking. The integration system is used for the testing and deployment of subdetector code updates, and thus provides the ability to quickly and smoothly deploy new code in case of changing detector conditions or upgrades. It also provides a backup of the production system in case of hardware failures. The development system is a replica of the production system, and is exclusively used for development and testing of the core infrastructure. It was used heavily and successfully e.g. for the work presented in this report.

The successful operational experience in 2010 revealed a number of small bottlenecks and shortcomings of the present system. One of the discovered weaknesses is the high CPU time consumption of the deserialization step, i.e. the unpacking of the incoming single event data in memory into the specific (ROOT-based) format required by the CMSSW event processing modules. The time spent in deserialization was found to scale roughly proportionally with the number of ROOT object data containers present in the event data. Consequently, for optimal data throughput, the number of root object containers should be kept to a minimum.

## 4. Improvements

Based on the system experience from data challenges and proton-proton collision runs in the years 2007 through 2009, a number of areas were identified for which improvements appeared desirable and feasible.

### *4.1. Robustness and Performance*

A high priority requirement for the DQM during data taking is the robustness (reliable operation) and error resilience. Errors may occur in the software in particular due to possible updates to the subdetector histogramming code, which are frequently done at times when detector or experiment conditions change. The goal of the development reported here was to improve the system response to failures of individual subsystem components and to accelerate the execution of the start-of-run and end-of-run state transitions.

A new configurable feature of the FUEP was adopted in which the actual CMSSW code is executed in a child process, forked from the FUEP master application. Communication and control of the child process with the master FUEP is handled using Unix signals. The master FUEP can detect child process failures, such as a crash of the DQM application, and can be configured to automatically restart the process after a specified period. Upon a stop or halt message, the master process waits for the child process to finish the event processing gracefully. If this fails within a specified period, the child process is terminated and transition finishes.

For the use of the master-child mode within DQM, a number of modifications to the FUEP applications, related to the DQM-specific input and output API described above, were necessary. In addition, modifications to the DQM run control function manager were implemented in order to improve the initialization performance and the reliability in case of failures of DQM components. The initialization time was significantly reduced by implementation of a parallel, i.e. asynchronous, instantiation of the full set of FUEPs, which was previously done in a serial, synchronous manner. The shutdown (stopping or halting) reliability was improved by implementing a timeout for the maximum duration for the termination of DQM components and the reporting of successful process completion to the central CMS run control.

### *4.2. Trigger Selection*

As described above, the SMPS is capable to request selected events based on specific trigger requirements. However, previous versions of the SMPS did not foresee complex logical

combinations of the trigger selection required by DQM. In the course of the improvements on the event server software, the selection facilities were upgraded to allow the full set of logical trigger bit combinations (comprised of AND, OR and NOT). Moreover, the application specific trigger conditions, when specified in a DQM consumer configuration, are dynamically passed through the SMPS up to the SM queue, this way minimizing maintenance overhead. However, the present system is limited to one set of trigger requirements per SMPS. In a future upgrade of the SMPS it is planned to implement a fully dynamic forwarding of several sets of trigger requirements from several DQM consumers through a single SMPS to the Storage Managers.

*4.3. Multi-Core Event Processing*

As described in section 3, the existing DQM system is limited to the single-threaded processing of a single event stream by each DQM consumer application. When event processing involves intensive calculations (e.g. tracking algorithms), the speed of the DQM application is generally CPU-bound, and only a subsample of events from the Storage Manager queue is processed for monitoring. This limitation is primarily imposed by the design of CMSSW event processing, which is done serially on the input event stream.

A way to achieve a higher rate was requested by several subsystems, as it would provide larger statistics for DQM results and increase sensitivity to rare events. In response to this situation an approach was developed to achieve parallelization by splitting the event stream to processes executed on multiple CPU-cores. It is sketched in figure 3 and described in the following.

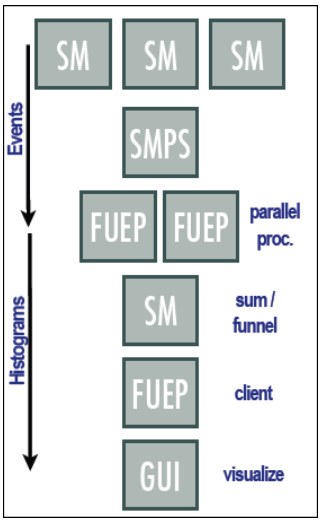


**Figure 3.** Schematic view of the setup for multi-core event processing as described in section 4.3.

Two models of parallelization were studied. In the first model multiple copies of the client process are created, which, instead of sending the histograms to the GUI, send them to a private instance of the SM which performs histogram aggregation and summing. A separate client collects summed histograms and send them to the GUI. This approach provides performance increase if the summing of histograms on the SM is fast in comparison to the event processing. This is the case for DQM applications with a small number of received histograms and histograms with a small number of bins. In the second model, a client is divided into reconstruction and histogramming process. The reconstruction step runs in multiple clients and the events get funneled through a SM to the final client, which analyzes the received data, creates histograms and sends the results to the GUI. The downside of this approach is the added

overhead of serialization and deserialization of event products, as they are transferred between processes. This approach provides performance increase for clients which perform expensive event reconstruction.

Both models required a few changes in the available infrastructure. The FUEP was enabled to fork multiple child processes that do parallel processing. The SMPS interface had to be modified to ship each event to only one client, so that no duplicate event processing occurs. To address the added deserialization cost, further improvements are being evaluated, including the reduction of the number of transferred products in an event, the aggregation of the data into a single product before the transfer, running the deserialization in parallel to the histogram processing code, or possibly using a radically multi-threaded parallel processing model in which the serialization is not necessary.

## 5. Conclusions

The CMS online DQM system has been in place and working robustly since the beginning of the data taking with beams at the LHC. Already in previous data challenges the system had been proven a cornerstone of CMS data taking efficiency and data certification. The main part of the system is operated in high rate spy-mode on event data retrieved from an event server downstream from the higher level trigger software filter (HLT). The spy-mode design choice was made in order to guarantee flexibility and robustness and to avoid interference or potential back pressure to the data acquisition system.

The whole DQM system is embedded in the online CMS data acquisition and run control environment, and re-uses a number of application components that were originally developed for and are in use in the HLT. A system of servers is used for the shipping of the event data into the DQM applications and of the histogram data to the visualization webserver.

For the operation in 2010 a number of improvements were implemented, described in this report. While the improvements in robustness and trigger selection of the events turned out to be a very significant success, the intent to gain performance improvements through parallel processing has not yet been realized. A number of tests carried out so far are presently being analyzed with the aim to find more suitable solutions.

## References

[1] Tuura L., Meyer A., Segoni I. and Della Ricca G., 2009, CHEP09, Computing in High Energy Physics, 2009 CMS data quality monitoring: systems and experiences Proc. (Prague, Czech Republic)
[2] CMS Collaboration, 1994, CERN/LHCC 94-38, Technical proposal, (CERN, Switzerland).
[3] Mommsen R., 2011, Proc. CHEP 2010, Computing in High Energy Physics, The Data Acquisition System of the CMS Experiment at LHC (Taipei, Taiwan)
[4] Tuura L., Eulisse G., Meyer A., 2009, Proc. CHEP09, Computing in High Energy Physics, CMS data quality monitoring web service (Prague, Czech Republic)
[5] Rapsevicius V. for the CMS DQM group, 2011, Proc. CHEP10, Computing in High Energy Physics, CMS data quality monitoring web service (Taipei, Taiwan)
[6] Brun R., Rademakers F., 1996, Proc. AIHENP96 Workshop, ROOT An Object Oriented Data Analysis Framework (Lausanne, Switzerland); see also http://root.cern.ch
[7] Bauer G., et al, 2007, Proc. CHEP07, Computing in High Energy Physics, The Run Control System of the CMS Experiment (Victoria B.C., Canada)
[8] Della Negra M., Petrilli A., Herve A., Foa L., 2006, CMS Physics Technical Design Report Volume I: Software and Detector Performance. (CERN, Switzerland)