CMS data quality monitoring web service

L Tuura[1], G Eulisse[1], A Meyer[2,3]
[1] Northeastern University, Boston, MA, USA
[2] DESY, Hamburg, Germany
[3] CERN, Geneva, Switzerland

E-mail: `lat@cern.ch, giulio.eulisse@cern.ch, andreas.meyer@cern.ch`

Abstract. A central component of the data quality monitoring system of the CMS experiment at the Large Hadron Collider is a web site for browsing data quality histograms. The production servers in data taking provide access to several hundred thousand histograms per run, both live in online as well as for up to several terabytes of archived histograms for the online data taking, Tier-0 prompt reconstruction, prompt calibration and analysis activities, for re-reconstruction at Tier-1s and for release validation. At the present usage level the servers currently handle in total around a million authenticated HTTP requests per day. We describe the main features and components of the system, our implementation for web-based interactive rendering, and the server design. We give an overview of the deployment and maintenance procedures. We discuss the main technical challenges and our solutions to them, with emphasis on functionality, long-term robustness and performance.

1. Overview

CMS [1] developed the DQM GUI, a web-based user interface for visualising data quality monitoring data for two reasons. For one, it became evident we would much prefer a web application over a local one [2, 3, 4] (Fig. 1). Secondly, we wanted a single customisable application capable of delivering visualisation for all the DQM needs in all of CMS, for all subsystems, for live data taking as much as archives and offline workflows [5].

Content is exposed as *workspaces* (Fig. 3) from high-level summaries to shift views to expert areas, including even a basic histogram style editor. Event display snapshots are also accessible. The server configuration specifies the available workspaces.

Within a workspace histograms can be organised into *layouts* to bundle related information together. A layout defines not only the composition, but can also provide documentation (Fig. 3(b), 3(e)), change visualisation settings, and for example turn the reference histogram display on (Fig. 3(d)). Shift views are usually defined as collections of layouts.

2. Implementation

Our server is built on CherryPy, a Python language web server framework [6]. The server configuration and the HTTP API are implemented in Python. The core functionality is in a C++ accelerator extension. The client is a GUI-in-a-browser, written entirely in JavaScript. It fetches content from the server with asynchronous calls, a technique known as AJAX [3, 4]. The server responds in JSON [3, 7]. The browser code forms the GUI by mapping the JSON structure to suitable HTML+CSS content (Fig. 2, 3).

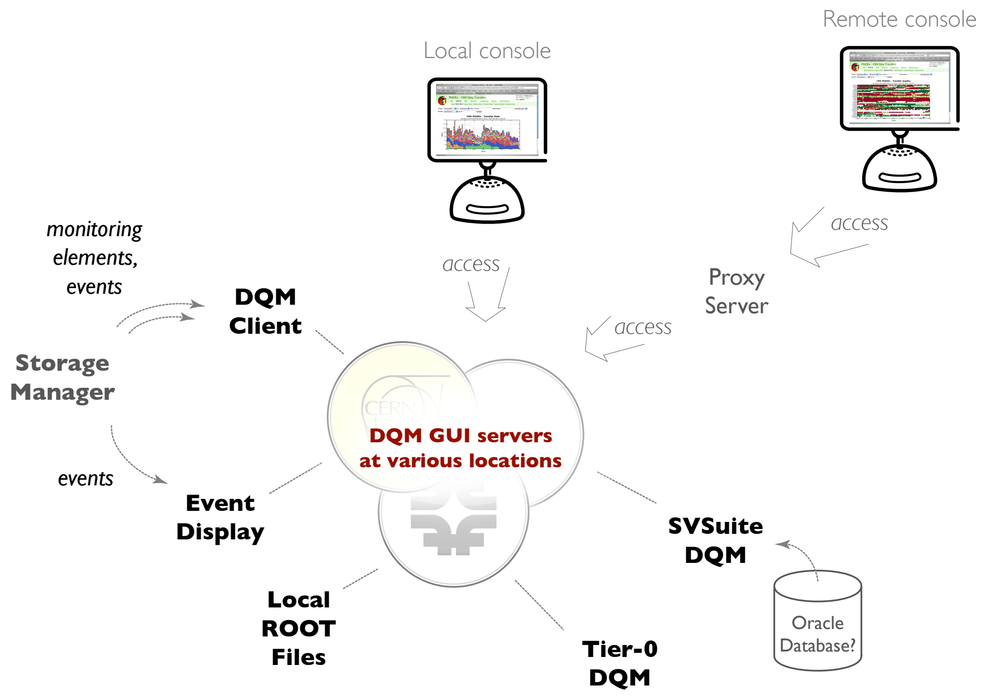


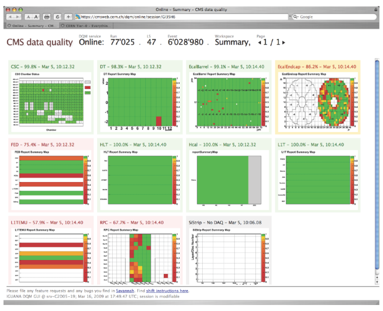
Figure 1. GUI architecture overview.

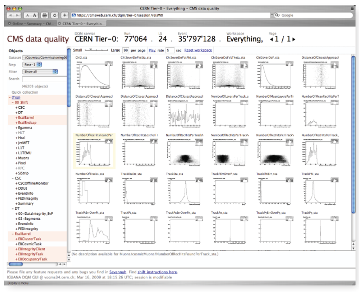
```
([{kind: 'AutoUpdate', interval: 300, stamp: 1237219783, serverTime: 96.78},
 { kind: 'DQMHeaderRow', run: "77'025", lumi: "47", event: "6'028'980",
   service: 'Online', workspace: 'Summary', page: 1, pages: 1, services: [...],
   workspaces: [{title: 'Summary', label: 'summary',
                category: 'Summaries', rank: 0}, ...],
   runs: ["Live", "77057", ...], runid: 77025},
 { kind: 'DQMQuality', items: [
   { label: "CSC", version:1236278233000000000,
     name: "CSC/EventInfo/reportSummaryMap",
     location: "archive/77025/Global/Online/ALL/Global run",
     reportSummary: "0.998", eventTimeStamp: "1236244352" },  ...]}])
```

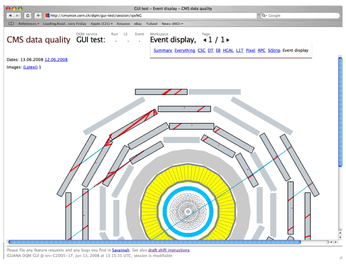
Figure 2. A JSON state response from which the Summary page of Fig. 3(a) was rendered. The response contains the minimal raw data needed for this particular page view, grouped by browser side JavaScript GUI plug-in which knows how to translate the state in HTML+CSS, including the user interaction controls.

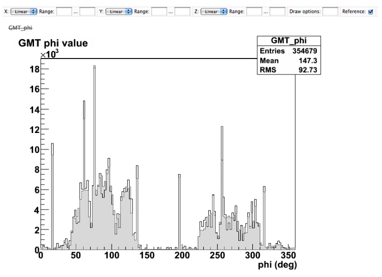
The user session state and application logic are held entirely on the web server; the browser application is "dumb." User's actions such as clicking on buttons are mapped directly to HTTP API calls such as `setRun?v=123`. The server responds to API calls by sending back a new full state in JSON, but only the minimum required to display the page at hand. The browser compares the new and the old states and updates the page incrementally. This arrangement trivially allows one to reload the web page, to copy and paste URLs, or to resume the session later.

The server responds to most requests directly from memory, yielding excellent response time.

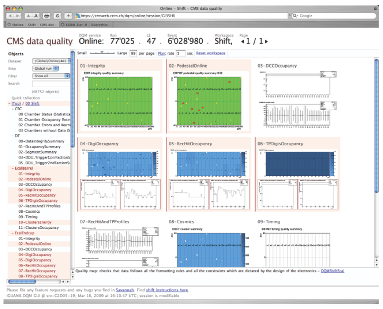

(a) General summary.


(b) Reduced histograms in normal browsing mode, rendered with anti-aliasing.


(c) An event display workspace.



(d) Reference histogram drawn.


(e) The standard ECAL barrel shift workspace.

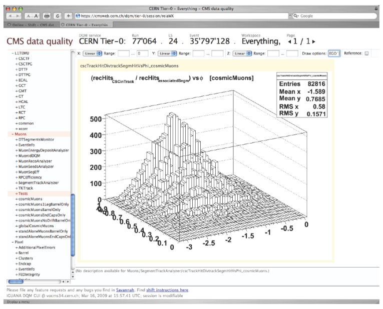

(f) Single histogram explorer and editor.

Figure 3. DQM GUI views.

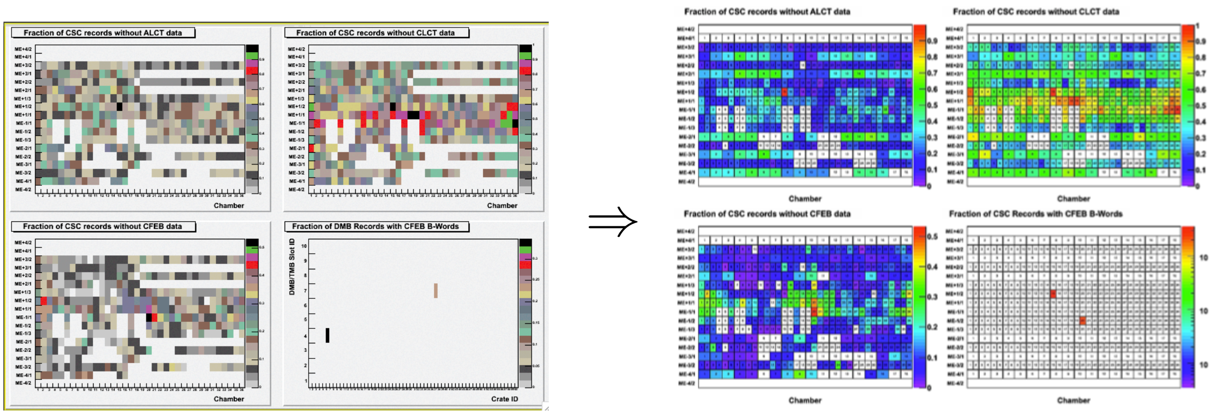


Figure 4. A render plug-in can be added to modify the appearance of a histogram.

All tasks which can be delayed are handled in background threads, such as receiving data from the network or flushing session state to disk. The server data structures support parallel traversal, permitting several HTTP requests to be processed concurrently.

The histograms are rendered in separate fortified processes to isolate the web server from ROOT's instability. The server communicates with the renderer via low-latency distributed shared memory. Live DQM data also resides in distributed shared memory (Fig. 6). Each producer hosts its own histograms and notifies the server about updates. The renderer retrieves

Figure 5. One of the central DQM and event display consoles at the CMS centre in Meyrin, 10 Sept 2008.

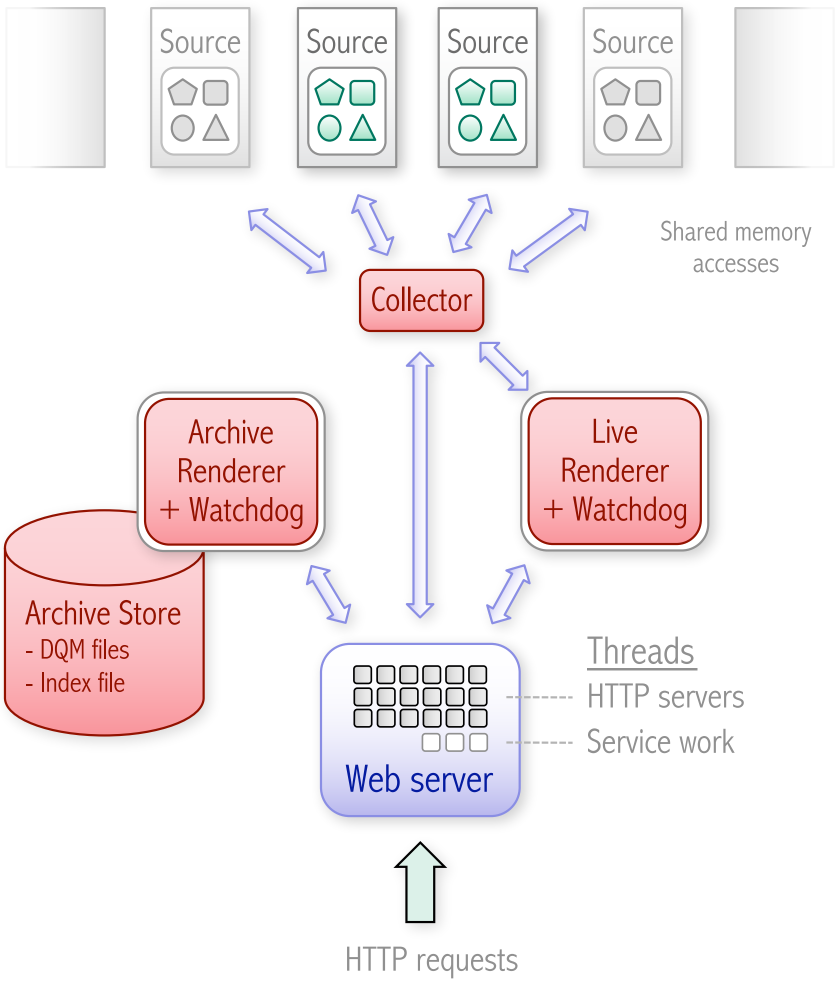


Figure 6. The distributed shared memory system.

histograms asynchronously from the producers on demand; although single-threaded for ROOT, it can have dozens of operations in progress concurrently. Recently accessed views are re-rendered automatically on histogram update to reduce the image delivery latency.

Our DQM data is archived in ROOT files [8]. As reading ROOT files in the web server itself would be too slow, use too much memory, prone to crash the server, and would seriously limit concurrency, we index the ROOT data files on upload. The GUI server computes its response using only the index. The ROOT files are accessed only to get the histograms for rendering in a separate process (Fig. 6). The index is currently a simple SQLite database [9].

The server supports setting basic render options, such as linear vs. log axis, axis bounds and ROOT draw options (Fig. 3(f)). These settings can be set interactively or as defaults in the subsystem layout definitions. The subsystems can further customise the default look and feel of the histograms by registering C++ render plug-ins, which are loaded on server start-up (Fig. 4). We improve image quality significantly by generating images in ROOT in much larger size than requested, then reducing the image to the final smaller size using a high-quality anti-alias filter.

3. Operation and experience

CMS centrally operates four DQM GUI instances for online and offline each, an instance per purpose for the existence of data: Tier-0, CAF, release validation, and so on. In addition at least four instances are operated by detector subsystems in online for exercises private to the detector. Most DQM developers also run a private GUI instance while testing. A picture of a live station is shown in Fig. 5.

Early on it became abundantly evident ROOT was neither robust nor suitable for long-running servers. Some three quarters of all the effort on the entire DQM GUI has gone into debugging ROOT instabilities and producing countermeasures. We are very pleased with the robustness of the rest of the DQM GUI system.

CMS typically creates approximately 50'000 histograms per run. The average GUI HTTP response time is around 270 ms (Fig. 7(a)), which we find satisfactory. The production server has scaled to about 750'000 HTTP requests per day with little apparent impact on the server response time (Fig. 7(b)). Interestingly the vast majority of the accesses are to the online production server from outside the online environment. This indicates the web-based monitoring and visualisation solution applies well to the practical needs of the experiment. We have exercised the GUI with up to 300'000 histograms per run. The GUI remains usable although there is a perceivable interaction delay. We plan to optimise the server further such that it has ample capacity to gracefully handle growing histogram archives and special studies with large numbers of monitored entities.

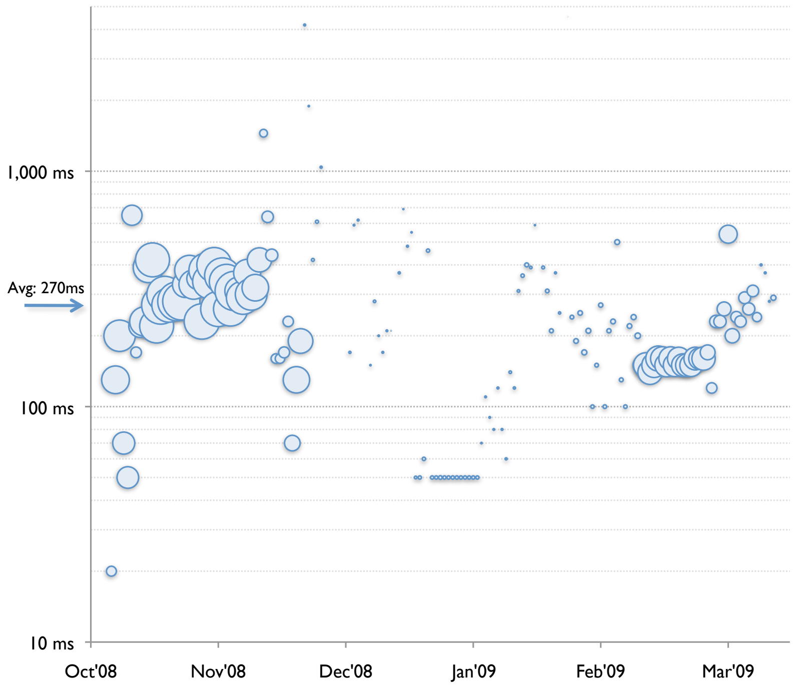
Acknowledgments

The authors thank the numerous members of CMS collaboration who provided abundant feedback and ideas on making the GUI server more effective and useful for the experiment.

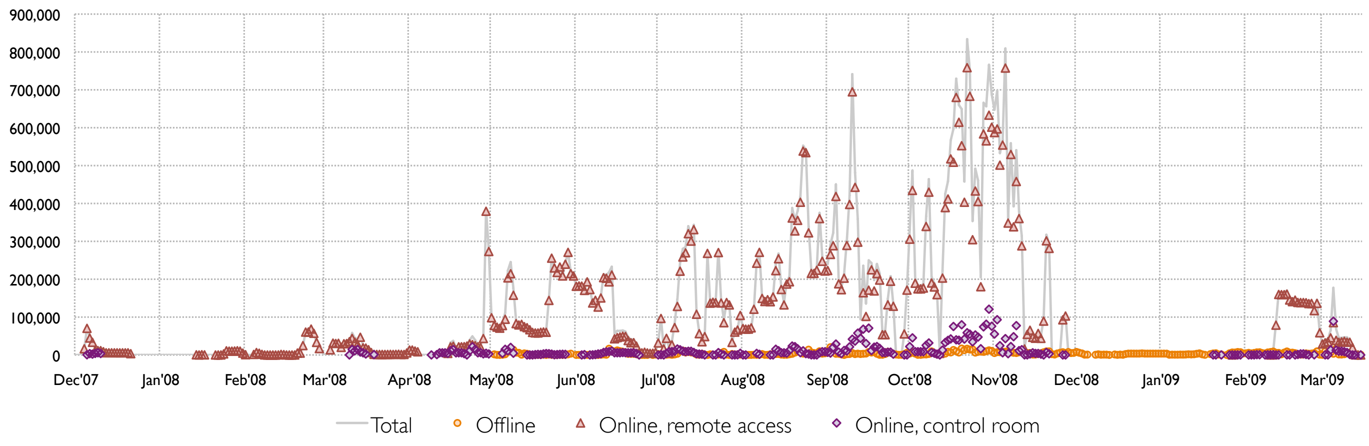
References

[1] CMS Collaboration, 1994, *CERN/LHCC 94-38*, "Technical proposal" (Geneva, Switzerland)
[2] Giordano D et al, 2007, *Proc. CHEP07, Computing in High Energy Physics*, "Data Quality Monitoring for the CMS Silicon Strip Tracker" (Victoria, Canada)
[3] Eulisse G, Alverson G, Muzaffar S, Osborne I, Taylor L and Tuura L, 2006, *Proc. CHEP06, Computing in High Energy Physics*, "Interactive Web-based Analysis Clients using AJAX: with examples for CMS, ROOT and GEANT4" (Mumbai, India)
[4] Metson S, Belforte S, Bockelman B, Dziedziniewicz K, Egeland R, Elmer P, Eulisse G, Evans D, Fanfani A, Feichtinger D, Kavka C, Kuznetsov V, van Lingen F, Newbold D, Tuura L and Wakefield S, 2007, *Proc. CHEP07, Computing in High Energy Physics*, "CMS Offline Web Tools" (Victoria, Canada)

[5] Tuura L, Meyer A, Segoni I and Della Ricca G, 2009, *Proc. CHEP09, Computing in High Energy Physics*, "CMS data quality monitoring: systems and experiences" (Prague, Czech Republic)
[6] CherryPy—A pythonic, object-oriented HTTP framework, 2009, `http://cherrypy.org`
[7] Introducing JSON, 2009, `http://json.org`
[8] ROOT—A data analysis framework, 2009, `http://root.cern.ch`
[9] SQLite—A library implementing a self-contained SQL database engine, 2009, `http://sqlite.org`



(a) The daily average DQM GUI response time versus date and number of requests per day.



(b) Number of DQM GUI HTTP requests per day from December 2007 to March 2009.

Figure 7. DQM GUI HTTP server performance.