The Data Quality Monitoring Software for the CMS experiment at the LHC

On behalf of the CMS Collaboration

Marco Rovere, CERN


Okinawa, Japan, April 13-17 2015
DQM is Everywhere
Preview

- Online DQM
- Offline DQM
- Generic Tools & Beyond Run2
  - further developments and possible improvements
Online DQM

Challenges and Solutions
It’s all about the Storage Manager Proxy Server

Online DQM during Run I

- Network DAQ to DQM communication relied on the Storage Manager Proxy Server (SMPS):
  - always served the latest event available
- Filtering capabilities provided by the SMPS
- HLT histograms merging provided by SMPS
- The SMPS disappeared in the improved DAQ2 design.
Taking responsibilities: give back to DQM what is DQM’s

- Completely file based and not event driven: receive all events/LS, should be processed sequentially
- No prior filtering applied
- HLT histograms merging in DQM hands
- Better separation of responsibilities
- Required deep redesign of processing logic.
Requirements

- Live event monitoring
- No external (XDAQ) process to start/stop DQM applications.
- No prior event selection
- Fast histogram merging

Solutions

- Reduce latency by:
  - Automatic file discovery
  - Tunable min. events/lumi
  - Skip to latest lumi/file.
- Reuse of HLT technology to handle all run transitions
- Event selection from within main framework
- Dedicated merging utility: fastHadd (ROOT+ ProtocolBuffer)

<table>
<thead>
<tr>
<th>Merging Utility</th>
<th>10 files</th>
<th>30 files</th>
<th>50 files</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROOT’s hadd</td>
<td>10.8s</td>
<td>48.9s</td>
<td>125s</td>
</tr>
<tr>
<td>fastHadd</td>
<td>3.8s</td>
<td>10.2s</td>
<td>17s</td>
</tr>
</tbody>
</table>
Offline DQM
Challenges and Solutions
Multi-threading in the CMS reconstruction framework (CMSSW)

**What is it?**

Ability to process many events in different threads.

**Where is it?**

At the root’s of CMSSW, affects everything.

**Why affects DQM?**

DQM uses a shared memory pool for holding ROOT histograms.

ROOT histograms.
Multithreading in DQM
Basic Concepts

- Serialize access to shared resources/data
  - Enforce policy via an interface

- Localize resources/histograms to each thread
  - Aggregate histograms at the end of data processing

- Hide complexity to end-users

- Avoid a full rewrite of the existing DQM Framework

- Prevent users from doing mistakes.
Transition to Multithreading in numbers

- Design of core components: ~6 man/months
- So far ~90% of the DQM and VALIDATION code has been migrated
- It involved 400+ classes and related helper ones
  - Several months(12+) of work
- The threaded DQM version is regularly exercised in the THREADED Integration Build (run twice per day)
- Comparison of single and multithreaded version are identical
- Performance numbers in Chris Jones’ talk [Track2 – 14/4/15 15:00]:
  - 5-15% loss in CPU usage
  - Factor of 3-4 in Memory reduction
    - Mostly not coming from DQM
Miscellanea and beyond Run2
Further Developments

• Data and Monte Carlo agreement key ingredient for many analysis
  • Embed this functionality into the DQM Framework

• DQMGUÍ has been constantly improved
  • ROOT6 compatible
  • New APIs to fully expose all its content
  • Automatic histogram stacking
  • Improved performance
Conclusions
Conclusions

• DQM framework proved to be extremely flexible and stable
  • DQM is used everywhere in CMS

• DQM Framework adapted and improved in the face of fundamental changes in the Online (DAQ2) and Offline (Multithreading) environments

• The central DQM Tools have been constantly improved during Long Shutdown 1.

• The new DQM design and implementation proved to be extremely effective since the beginning of the commissioning period in 2015.
BACKUP SLIDES
DQM in CMS: Core Components

- **DQMStore**: shared container that holds all Monitoring Information.
- **MonitorElement**: 
  - ROOT objects
  - Quality Information
  - Folder hierarchy
  - Flags
- **DQMNet**: layer to ship monitoring information over network.
- **DQMService**: ties **DQMStore** and **DQMNet** together.

--- CMSSW Services

----- Standalone C++
MultiThreaded DQM

Book Interface
Serialized by internal lock

Module A Thread 1
Events X+N

Module A Thread N
Events Y+N

Booking

Filling

Aggregation

DQM Internal Memory

Persist/Writing

Disk
DQM GUI

A central component of the data quality monitoring system of the CMS Experiment is a web site for browsing data quality histograms. It guarantees authenticated Worldwide access. It is a single customizable application capable of delivering visualization for all the DQM needs in all of CMS, for all subsystems, for live data taking as much as archives and offline workflows.
DQM GUI Architecture: C++, CherryPy, JS

Render Plugins

Collector

Source

Collector

Renderer + Watchdog

Renderer + Watchdog

Archive Store - Index file

Web server

Threads

HTTP servers

Service work

Shared memory accesses

HTTP requests
DQM GUI Developments during LS: APIs

The DQM GUI is capable of exposing its index via many APIs.
Performance Plot of DQM GUI

HTTP Requests Served Per Day - Online Server Offsite

HTTP Response Time – Daily Average - Offline Server
DQM (GUI) in numbers

Show some interesting statistics:

✓ O(15K) lines of C++ code (web server accelerator, render engine, index manipulation), O(5K) python lines and O(4K) javascript lines.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Offline Server</th>
<th>RelVal Server</th>
<th>Online Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAMPLES</td>
<td>264.383</td>
<td>31.587</td>
<td>16.042</td>
</tr>
<tr>
<td>SOURCE FILES</td>
<td>303.580</td>
<td>31.744</td>
<td>387.298</td>
</tr>
<tr>
<td>DATASETS</td>
<td>2.068</td>
<td>18.379</td>
<td>1</td>
</tr>
<tr>
<td>CMSSW VERSIONS</td>
<td>--</td>
<td>90</td>
<td>--</td>
</tr>
<tr>
<td>UNIQUE OBJECTS (MEs)</td>
<td>913.174</td>
<td>1.649.086</td>
<td>346.448</td>
</tr>
<tr>
<td>STREAMERS</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>DISK SPACE (TB)</td>
<td>2.5</td>
<td>0.38</td>
<td>0.06</td>
</tr>
</tbody>
</table>
Data vs Monte Carlo automatic tools

• Data and Monte Carlo agreement key ingredient for many analysis
  • Embed this functionality into the DQM Framework
• Properly combined many MC samples
  • event weight applied separately for each MC sample
  • The scaling factor is the cross section taken as a parameter from the configuration. The number of produced events is taken from an existing ME
• Datasets are summed in a final, cumulative, sample
• DQMGUI has the capability to properly stack histograms from different sample.
DQM in CMS: Requirements

 ✓ DQM should aggregate event level information that is sensitive to both detector and software (HLT) problems in one central place (web server).

 ✓ Fast turn-around and small latency. Update of info frequently during runs. Quality tests up to every lumi-section.

 ✓ Automatic alarms should notify about problems.

 ✓ Synoptic overview of detector status (front page).

 ✓ Shift level histograms.

 ✓ Expert histograms.

 ✓ The DQM information is key input to the creation of the good run list (aka JSON).

 ✓ The web server should be accessible everywhere, not just inside P5.

 ✓ DQM should be maintainable in a modular way by subsystems with fast updates outside regular release cycles.

 ✓ DQM needs to run in spy mode, in order to not interfere with the data taking.
DQM Offline

Prompt Reconstruction, Calibrations, re-reconstruction, simulation and release validation all use the same processing model.

Histograms (ME) created in jobs, saved in normal data files, harvested periodically and merged into full statistics with DAQ, DCS info and finally tested for quality and summarised.

Resulting histograms are uploaded to the GUI web server hosted at CERN, backed up to CASTOR/EOS.

Final quality summary flags are stored into condition database for certification.

Differences are in content and timing. Tier-0, Tier-1s re-determine detector status using full event statistics, full reconstruction, plus add monitoring for physics objects; Tier-0 $\Delta t \sim 48h$, Tier-1s days+. CAF $\Delta t$ hours to days on Al-Ca entities. Validation verifies MC data.
CMSSW Multi-{Core, Thread}

Global

✓ Sees transitions on a ‘global’ scale
  ✓ see begin of Run and begin of Lumi when source first reads them
  ✓ sees end of Run and end of Lumi once all processing has finished for them

✓ Multiple transitions can be running concurrently
  ✓ two or more begin or end Runs (for different runs)
  ✓ two or more begin or end Lumis
  ✓ and end can be occurring while another begin is running

Events are not seen ‘globally’

Stream

✓ Processes transitions serially: begin run, begin lumi, events, end lumi, end run.

✓ Multiple streams can be running concurrently each with own events.
Each module, in turn, sees a fixed sequence of events raised by the framework. Each module books the MonitorElements into the central DQMStore and receives back pointers to the newly created MEs. Events are serially processed and the MonitorElements are properly filled. At every End{Lumi,Run}, the corresponding MonitorElements are permanently written on disk. In the case in which the DQMNet is available, all modified MonitorElements will push their changes to the other side of the network channel (GUI’s collector). The harvesting step is similar in concept, but will only see Run and Lumi transitions and will only have access to MonitorElements, not to the events. Quality tests are run in the harvesting in the end* transitions, depending on their configuration.