Summer Student Report: DQM optimization

Vojtech Simetka
vojta.simetka@gmail.com
CERN European Organization for Nuclear Research
Meyrin, Switzerland

Introduction

The Data Quality Monitoring (DQM) Software proved to be a central tool in the CMS experiment. Its flexibility allowed its integration in several environments: Online, for real-time detector monitoring; Offline, for the final, fine-grained Data Certification; Release Validation, to constantly validate the functionality and the performance of the reconstruction software; in Monte Carlo productions [1, 2, 3]. The resources used by the different DQM software systems, both in terms of time and memory, are ever-growing. Having in mind the next to come LHC phases, the improvement of the performance of the DQM and Validation code will make possible the validation of extreme condition scenarios (events with high number of pileup vertices).

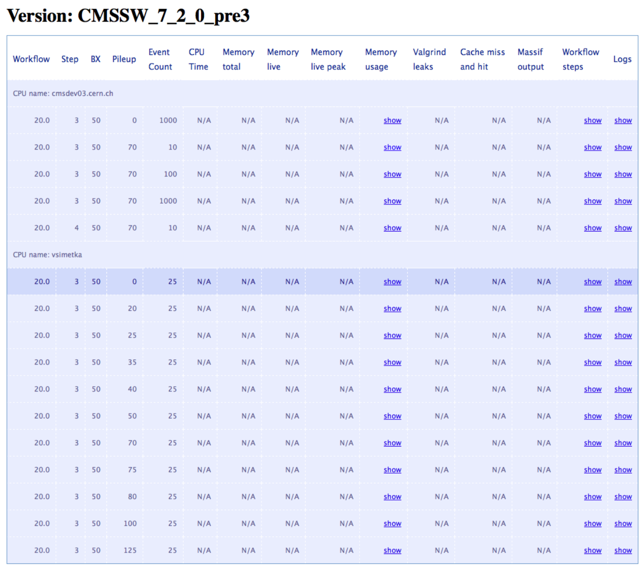
Goals

The goal of this project is to profile the software, identify the most problematic points, analyse them and find the most cost-effective solutions.

Profiling

In order to optimise a software, one has to have a good understanding of the source code. Even then it can be quite difficult to find what is the problem in terms of performance and what can be improved. However, we can use profilers to run the software and see where the inefficiency is. We have used the IgProf profiler [7], which is closely bound with the CMSSW, to profile the performance and memory of the DQM offline workflows used for release validation. The Valgrind tools [8] (namely memcheck, cachegrind and massif) were used to give us even better understanding of memory allocation, cache usage and possible memory leaks. In order to visualize how the live memory looks like during the run, we have sampled the process with linux ps command.

Because we are running standard workflows used for release validation, we have automatized the profiling process by creating easy to use interface. The resulting tool runs all the profilers with appropriate input and parameters, analyses the output and stores it to a webpage. It allows us to easily modify workflow configuration by changing pileup, bunch crossing time spacing and number of events. The tool automatically detects version of the CMSSW being run and the machine used to profile. This is important because the profiling results are machine dependant. All the data created can be browsed on output webpage (see figure 1). This tool will be part of the DQM software and made available for all developers.



Version: CMSSW_7_2_0_pre3

Workflow	Step	BX	Pileup	Event Count	CPU Time	Memory total	Memory live	Memory live peak	Memory usage	Valgrind leaks	Cache miss and hit	Massif output	Workflow steps	Logs
CPU name: cmsdev03.cern.ch														
20.0	3	50	0	1000	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	70	10	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	70	100	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	70	1000	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	4	50	70	10	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
CPU name: vsimetka														
20.0	3	50	0	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	20	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	25	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	35	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	40	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	50	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	70	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	75	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	80	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	100	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show
20.0	3	50	125	25	N/A	N/A	N/A	N/A	show	N/A	N/A	N/A	show	show

Figure 1: Output webpage.

The output of the profiling tools showed us several modules that can be improved. We analysed both memory and time reports and found several candidates for improvement especially in processing time.

Results

Profiling of the workflow 20.0 (Single Muon Gun with Transverse Momentum of 10GeV) step 3 (Reconstruction + DQM + Validation) with pileup 70, beam crossing time spacing 50 ns and 10 events showed us the biggest offenders of the validation step:

Module	CPU Time before	CPU Time after
MultiTrackValidator	297 s	179 s
RecoMuonValidator	18 s	16 s
MuonTrackValidator	17 s	16 s.
SiStripMonitorTrack	13 s	4 s

The changes we made reduced the processing time by 13.2% and the total memory needed for running the whole step 3 of workflow 20.0 by 12.1% (see figure 2). When we split the step into the substeps, the improvement in the Validation substep (which we focused on) is 26% (figure 3).

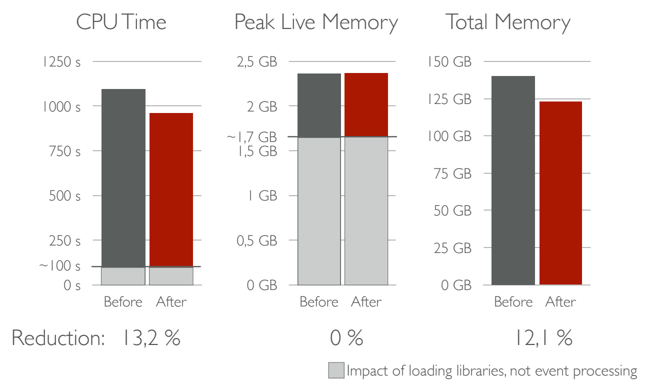


Figure 2: CPU time, live memory and total memory of step3 before and after our changes.

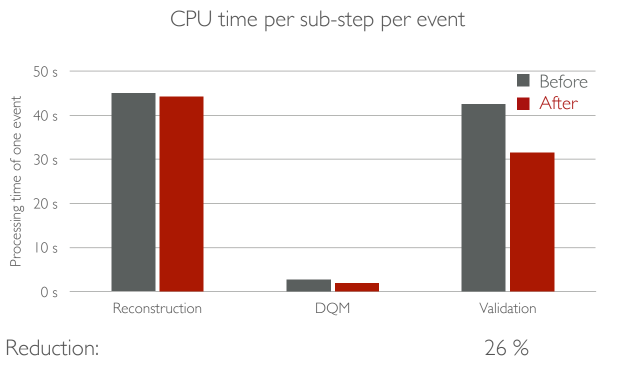


Figure 3: CPU time improvement of step3 before and after our changes.

To see how the DQM software deals with higher pileup scenarios we conducted a pileup study. The conclusions of the study are, that with increase in pileup, the time and memory increase is polynomial of second order while the live memory increase is linear (figure 4).

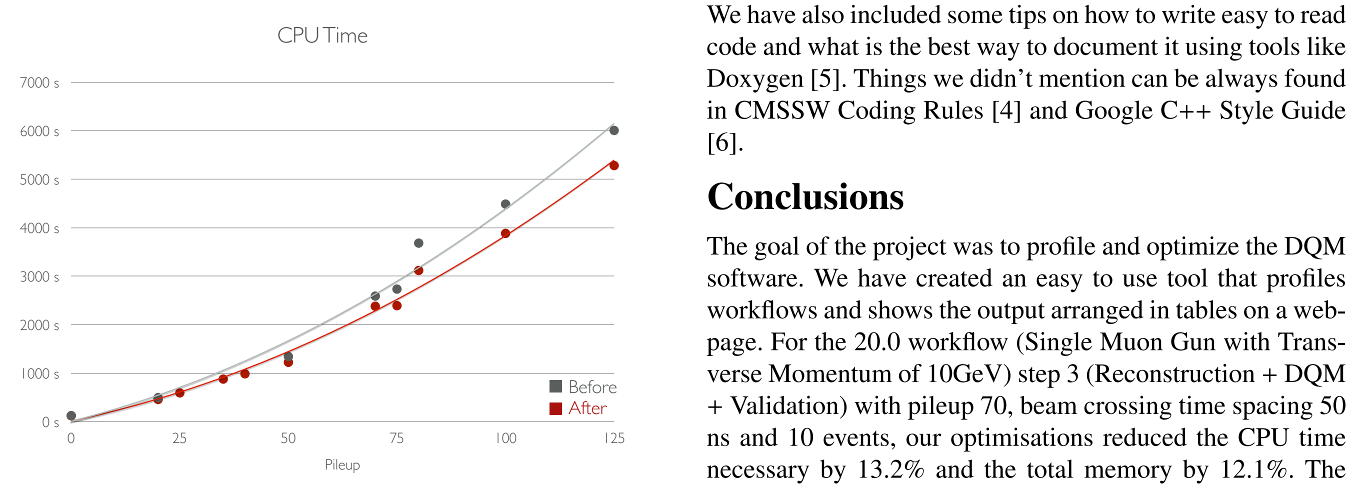


Figure 4: Pileup influence on CPU time, total memory and live memory.

Good Practices

During the optimisation, we have often encountered similar inefficient code. In order to improve software development, we have created a list of good practices for DQM developers. It covers topics like matching collections, improving conditions evaluation by reducing number of evaluated functions, memory allocation practices, improving cache hit ratio etc. We have also included some tips on how to write easy to read code and what is the best way to document it using tools like Doxygen [5]. Things we didn't mention can be always found in CMSSW Coding Rules [4] and Google C++ Style Guide [6].

Conclusions

The goal of the project was to profile and optimize the DQM software. We have created an easy to use tool that profiles workflows and shows the output arranged in tables on a webpage. For the 20.0 workflow (Single Muon Gun with Transverse Momentum of 10GeV) step 3 (Reconstruction + DQM + Validation) with pileup 70, beam crossing time spacing 50 ns and 10 events, our optimisations reduced the CPU time necessary by 13.2% and the total memory by 12.1%. The CPU time of the Validation substep alone was reduced by

26%. A list of good practices for both DQM and C++ development was created to improve the DQM software quality. The project improved my knowledge in multiple disciplines: computer science (profiling and code optimization, Linux tools, git, etc.) physics and team collaboration.

References

[1] A. Batinkov. *DQM poster*. Seoul, Korea, 2013.

[2] A. Batinkov, M. Rovere, L. Borrello, F. D. Guio, D. Duggan, and S. D. Guida. *The CMS Data Quality Monitoring Software: Experience and Future Improvements*, 2013.

[3] F. D. Guio. The compact muon solenoid experiment. *CHEP2013 Computing in High Energy Physics 2013*, 2013.

[4] WWW pages. cmmsw/DocumentationCodingRules at CMSSW_7_2_X . cms-sw/cmssw [online]. `https://github.com/cms-sw/cmssw/tree/CMSSW_7_2_X/Documentation/CodingRules`, [cit. 2014-08-28].

[5] WWW pages. Doxygen: Main Page [online]. `http://www.stack.nl/ dimitri/doxygen/`, [cit. 2014-08-28].

[6] WWW pages. Google C++ Style Guide [online]. `http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml`, [cit. 2014-08-28].

[7] WWW pages. IgProf, the Ignominous Profiler [online]. `http://igprof.org`, [cit. 2014-08-28].

[8] WWW pages. Valgrind Home [online]. `http://valgrind.org`, [cit. 2014-08-28].