Understanding IO patterns and performance of CMS Data Analysis across T2s worldwide

Leonardo Sala for the CMS Collaboration

Abstract

We discuss the IO patterns and performance of CMS Data Analysis based on measurements on "the bench" as well as operational observations during the initial LHC data taking period. It was observed the default I/O settings for CMS data analysis resulted in poor CPU efficiency and high I/O load at many T2 sites. This effort developed tools to measure CMSSW I/O performance for grid-based analysis. Our work enabled CMSSW developers to understand how new I/O strategies work across the full range of deployed T2 storage solutions. We discuss our results, based upon aggregate I/O statistics collected worldwide.

Presented at CHEP2010: International Conference on Computing in High Energy and Nuclear Physics 2010
Understanding I/O patterns and performance of CMS Data Analysis across T2s worldwide

Leonardo Sala  
on behalf of the CMS Collaboration  
E-mail: leonardo.sala@cern.ch

Abstract. We discuss the I/O patterns and performance of CMS Data Analysis based on measurements on "the bench" as well as operational observations during the initial LHC data taking period. It was observed the default I/O settings for CMS data analysis resulted in poor CPU efficiency and high I/O load at many T2 sites. This effort developed tools to measure CMSSW I/O performance for grid-based analysis. Our work enabled CMSSW developers to understand how new I/O strategies work across the full range of deployed T2 storage solutions. We discuss our results, based upon aggregate statistics collected worldwide.

1. Introduction
The recent success of first LHC operations also depends on the computing infrastructure provided by the WLCG. The WLCG infrastructure allows the deployment of a coherent common computational infrastructure over an inhomogeneous set of processing and storage technologies. This approach simplifies the usage of the resources by the physics community and also allows to access to different technologies with a common layout; to reproduce the same in a single computational center, a not negligible amount of hardware and personnel should be involved. The understanding of performances of the CMS software CMSSW [1] on different storage technologies can profit from this common access layout.

The optimization of I/O operations as reading, seeking files, is truly important in order to have faster analysis jobs and a lighter load on the computational sites. Nevertheless, given the sometimes really different philosophies lying under the various filesystems and storage technologies, it is indeed fundamental to test the improved code on a wide range of sites, as unexpected side effects could rise.

The drawback in using the Grid for performing tests is simply the other side of having a distributed computing infrastructure: the tester does not have the control on testing environment, and the measurements performed can be severely biased by various factors (site overload, high CPU demanding jobs et cetera), while on a local site it is possible to profit of dedicated hardware for testing. These considerations lead to careful thinking of how the tests should be performed, and how to control the possible biases.

2. The TestBed: CMSSW Performance Toolkit (CPT)
The necessary ingredients for performing the tests are:

- Some dedicated CMSSW analysis code and configuration filesystem
- A set of quantities to be measured
- A tool to easily analyze the results

Two different analysis codes has been chosen to exploit different I/O regimes: the first one is similar to a standard CMSSW physics analysis, while the second one exploits an higher limit to I/O access with a low CPU intensive job. The first task is performed through a simple dedicated code (called MiniTRee v.3, or MTR3), which reads 42 branches (containing most of the physics objects related to analyses) from CMS standard data files and performs some basic tasks, such as computation of invariant masses; on the other hand, the higher limit is set by an example code provided by the CMS JetMET group, called JetPlotExample or JPE, which simply plots 3 quantities without performing any computation.

Information on job performance is provided by various sources. CMSSW reports CPU and storage access statistics in an XML file, which contains a wide range of variables to be analyzed. The CMS tool job Grid job submission, CRAB (CMS Remote Analysis Builder [3]), also produces some job statistics, as CPU efficiency and job Grid latency. All this information is job-related, but it does not give an handle on the environment under the which this is executed (e.g. worker node load, network traffic). This is necessary in order to understand the bias which underlies the measurement. Consequently CPU, disk and network measurements are done using bash scripts relying on the `vmstat` utility and `/proc/dev/net`.

All this information needs to be handled and analyzed in an efficient way: the CMSSW Performance Toolkit (CPT) has been created to hit this target. CPT is made by two main scripts coded in a plug-in like structure (Fig. 1), written in Python and using ROOT python bindings (PyROOT [2]):

- `cpt_getJobInfo.py`: this script parses the log files provided by the job and translate all the variables in Python dictionaries, which are then automatically saved in ROOT files. Plugins are provided for each information source (at the moment, CMSSW, CRAB, SYS), making this tool easily adaptable also to non-CMS applications;
- `cpt_getStats.py`: an highly-configurable tool which plots and compares the selected distributions and measurement sets. It furthermore produces twiki-formatted tables with statistical quantities (average mean, standard deviation) for the selected quantities.

A job script is also part of CPT as an example of how retrieve the information from `vmstat` and make it available to the toolkit.

![Figure 1. The structure of CPT.](image.png)

The input for `cpt_getJobInfo.py` are the various job logfiles, processed by the proper plugin; its output then can be used as input for `cpt_getStats.py`, in order to analyze the results.
3. Test description
CMSSW data reading process can be configured in various ways, allowing to choose different methods for different usecases and storage technologies. Currently, there are three available mechanisms:

- application-only: ROOT will do the caching. If a non-zero cache is given, a TTreeCache of that size will be created per open file. Asynchronous read-ahead will be turned off and the cache will be filled with normal reads;
- storage-only: ROOT will drive the caching using a prefetch list, but will not allocate a cache of its own. If a non-zero cache is given, a TTreeCache with a read-list of that size will be created, but no actual cache buffer;
- lazy-download: remote files will be downloaded to a local shadow file on demand in 128MB segments. ROOT reads will be directed to this local file. On local filesystems (or equivalent, as Lustre), this behavior is switched to storage-only.

The tests performed during April 2010 were aimed to both measure the performances of the CMSSW version available at that time and a set of patches to ROOT developed by CMSSW developers in collaboration with ROOT developers.

By default, CMSSW reads all the branches from the data files, and this was triggering a bug in the TTreeCache implementation in ROOT 5.22\(^1\) (the release linked to CMSSW software available at that time), which caused a factor 2 of more data read with the respect of the actually requested. Patches were aimed to solve this behavior, and to include a read-coalesce read mechanism, where a set of small-sized reading operations are merged in a single bigger one.

The modified CMSSW code is supposed to improve at least the application-only setting as compared with the lazy-download one, that usually has better performances but it is more stressing for the network. One site for each storage technology has been chosen: T2_ES CIEMAT (dCache [4]), T2_FR_GRIF LLR (DPM [5]), T2_IT_Bari (Lustre [6]+StoRM [7]), T2_US_Nebraska (Hadoop [8]+BeStMan [9]). Local tests on a dedicated machine available at T3_CH_PSI (dCache) have been used as control checks.

The environment load has been measured by the insertion of a 60 seconds sleep command before the actual job execution: this allowed to record the CPU, disk and network load, and to discard too biased measurements. Tests ran daily, and the best performance for each test have been chosen for each site, also considering the load on the executing machine as described above.

4. Discussion of results
Preliminary tests performed on the dedicated setup at T3.CH.PSI revealed that the patches can reduce the network load by a factor 5, with a drastic reduction of the execution time. This can be seen in Fig. 2, where the differences in the x-axis for the three curves are due to the different execution times, revealing an improvement also in overall performance beside the reduction of network load.

Lazy-download in general performs very badly when the amount of needed data is much less than the total (the JPE job, Fig. 3). In this case the overhead is not negligible and leads towards an efficiency loss. A different situation applies to Bari, where due the Lustre filesystem the actual reading is done using the storage-only mode: in this way, CMSSW can fully take profit of the parallel filesystem. Using an analysis-like job like MTR3, lazy-download performs usually better than the default application-only setting (Fig.4): nevertheless, after applying the patches the performance of this latter greatly improves (up to a factor 2) and performs equally or better than lazy-download, but with a much smaller network load. It can be seen that the

\(^1\) To get an insight of ROOT recent I/O developments, see Philippe Canal’s contribution in these proceedings.
necessary reading time is reduced on most types of storage technologies (Fig. 5). Also in this case, the Lustre filesystem holds the best performance.

5. Conclusions
The tests performed showed that the modified CMSSW version holds improved performances in the I/O operations, which is directly translated into better CPU efficiency and a lower execution time. The recorded performances using cached TTree files are comparable or better than the ones without lazy-download settings. Another result found is the outstanding performance of the
Lustre filesystem, which can be a factor 2 or more better than the other settings tested. As a consequence of these tests, the patches described in the previous paragraph have been included in official CMSSW releases.

6. Acknowledgments
Thanks to Brian Bockelman (the actual developer of the CMSSW patches) and Lassi Tuura for their patience and explanations. Many thanks also to the administrators of the tested sites (Andrea Sartirana, Derek Feichtinger, Giacinto Donvito, Jose Hernandez) for their helpfulness and support and to the CMS Analysis Operation group, under which this work has been done.

References
[1] https://twiki.cern.ch/twiki/bin/view/CMS/SWGuide
[2] R. Brun and F. Rademakers, ROOT - An Object Oriented Data Analysis Framework, Proceedings AIIHENP'96 Workshop, Lausanne, Sep. 1996, Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86. See also http://root.cern.ch
[3] D. Spiga et al., Nucl.Phys.Proc.Suppl.177-178:267-268 (2008)
[4] http://www.dcache.org
[5] https://twiki.cern.ch/twiki/bin/view/LCG/DataManagementDocumentation
[6] http://wiki.lustre.org and the contribution “Lustre filesystem for CMS Storage Element (SE)” in these proceedings
[7] http://storm.forge.cnaf.infn.it
[8] http://hadoop.apache.org/
[9] https://sdm.lbl.gov/bestman/