



DESY Summer Student programme report

Access to CMS data - analyze and optimize

Andrzej Wronka^{a *}

a) AGH University of Science and Technology, Marian Miesowicz Faculty of Physics
Reymonta 20, 30-059 Krakow, Poland

September 9, 2010

*andrzejwronka@gmail.com

Acknowledgments

My sincere thanks are due to my supervisor, Hartmut Stadie, who supported me in my task and made that work in DESY was an interesting experience that I'll always be happy to be recalled. Also I would like to thank Christoph Wissing, with whom I had the pleasure to cooperate. His professionalism and commitment gave meaning to my work and helped in the achievement of the objective.

1 Introduction

This paper contains a report on the work done during the DESY Summer Student programme in the year 2010. The presented results are part of the analysis of the Tier's 2 performances made for the CMS Computing Project.

1.1 CMS experiment

The Compact Muon Solenoid (CMS) experiment is designed as a general-purpose particle physics detector built at the Large Hadron Collider (LHC) at CERN to observe a wide range of particles and phenomena produced in high-energy $\sqrt{s}=14\text{ TeV}$ proton-proton collisions.

1.2 The Computing Project

The CMS Computing is based on Grid technologies and is organized in a so called Tier structure, with one Tier-0 at CERN at the location of the experiment, 7 Tier-1 centers and 40 Tier-2 centers round the world. DESY is hosting one of the Tier-2 centers, which is operated in close collaboration with the DESY-IT staff and CMS members from the University of Hamburg. The CMS Tier-2 centers are used for centrally managed production of Monte Carlo events and analysis of CMS data by individual users.

To complement the Grid based computing and storage resources DESY runs the National Analysis Facility (NAF) for the German user community. The basic structure of NAF is shown in Fig. 1.

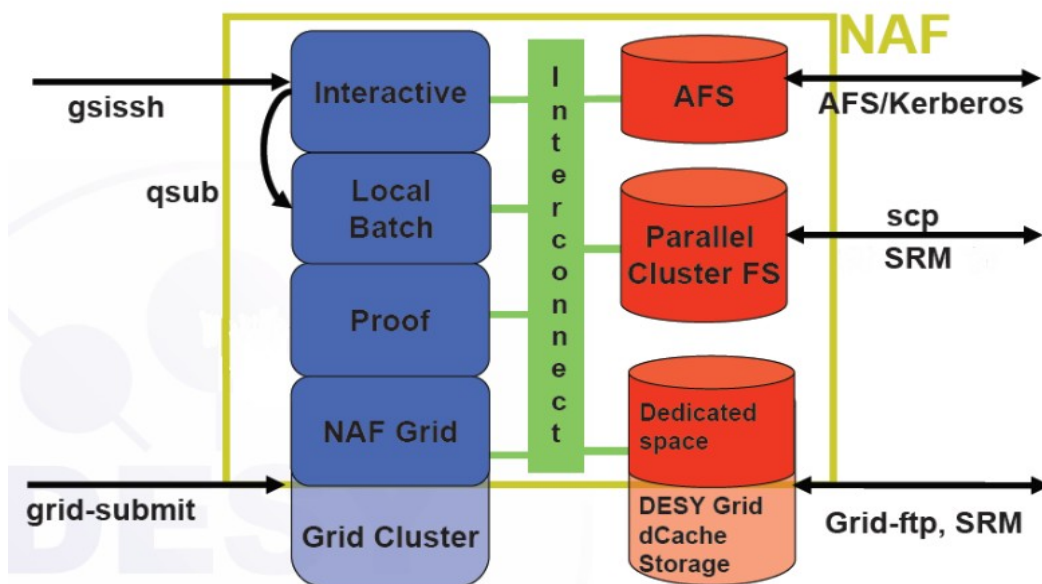


Fig. 1. : NAF overview.

2 Data analysis

Performance tests of analyzing and storage data in computer grid are significant for each HEP experiment. We are always looking for the best configuration of the computer storage system to make the analysis more efficient.

My project concerning data analysis of DESY's Tier's 2 performances. Specially I focused on retrieving parameters of *NAF-GRID - DESY-GRID* (Fig. 1) transfer. I worked on CMSSW (CMS SoftWare) version 3.6.1 and used glite32 UI.

2.1 CMS data and configuration

I have worked on data from CMS Dataset Bookkeeping System (DBS), which is a database and user API that indexes event-data data for the CMS Collaboration.

Used dataset: */Mu/Run2010A-PromptReco-v4/RECO*

Main tool used by me in analysis was CRAB (CMS Remote Analysis Builder). CRAB provides a utility for creating and submitting CMS jobs for input datasets. Fig. 2 shows an overview schema of CRAB working. Jobs were run with the configuration (*crab.cfg*):

Scheduler: SGU (Sun Grid Engine)

Lumi mask: Cert_132440141961_7TeV_StreamExpress_Collisions10_JSON.txt

Lumis per job: 20

Parameter set file: patTuple_standard_cfg.py

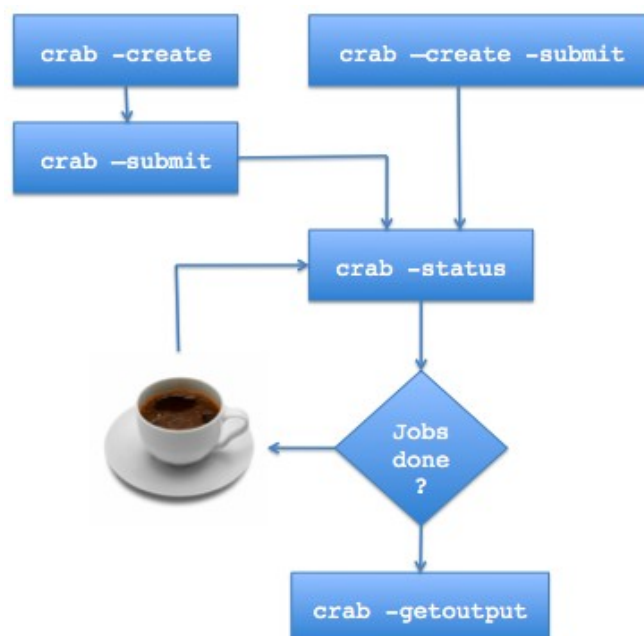


Fig 2. : Basic workflow of CRAB.

2.2 Physics analysis

As output of CRAB. PAT Tuple ROOT files are created. To produce histograms with physics analysis, I have used FWLiteAnalyzer. Plots are contained in *analyzePatBasics.root* files. We could retrieve useful statistics only the sample is large, so usually it is necessary to do 'summary' histograms - we need merge plots for all jobs set. For this purpose we have used the program **hadd**:

hadd analyzeAll.root @out ,

where @out is a list with names of output root files and *analyzeAll.root* will contain the merged histograms.

An example plot is shown the di-muon invariant mass spectrum. It was drawn for 253 crab's jobs, which corresponds to more than 500k events.

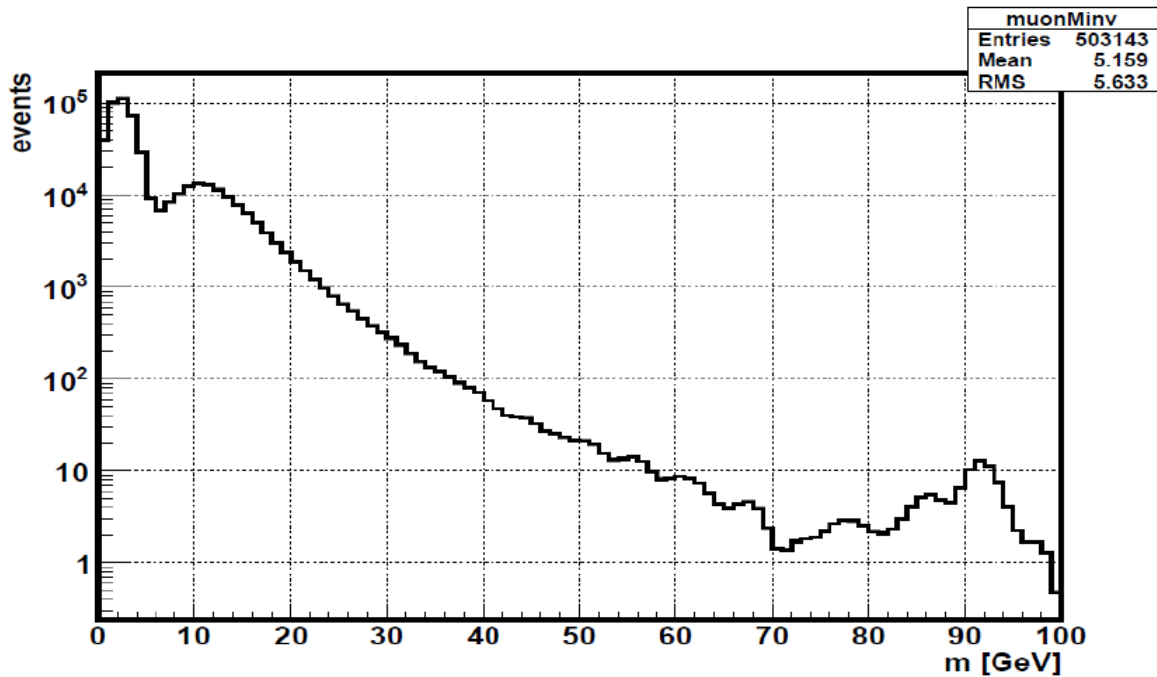


Fig. 3. : dimuon inv mass.

In Fig. 3 we can notice peak around 92 GeV, which corresponds to $\mu^+ \mu^-$ pairs from to Z - decay.

2.3. Framework job reports analysis

In addition to ROOT files, CRAB produces as output Framework Job Reports (*FJR*), which contain useful storage and performance statistics. To extract the information it is necessary to parse the FJR files (*I wrote script in python*) – these are XML documents.

The following variables were studied:

- exetime - running time of the job,
- readtime - time spent for reading data,
- size of read data,
- number of events,
- operation number - number of times the operation was attempted,
- maximum and minimum readtime - the smallest and the biggest time an operation took,
- number of dcap operations – number of reads from the dCache area.

Fig. 4 shows an example histogram with amount of data read per event.

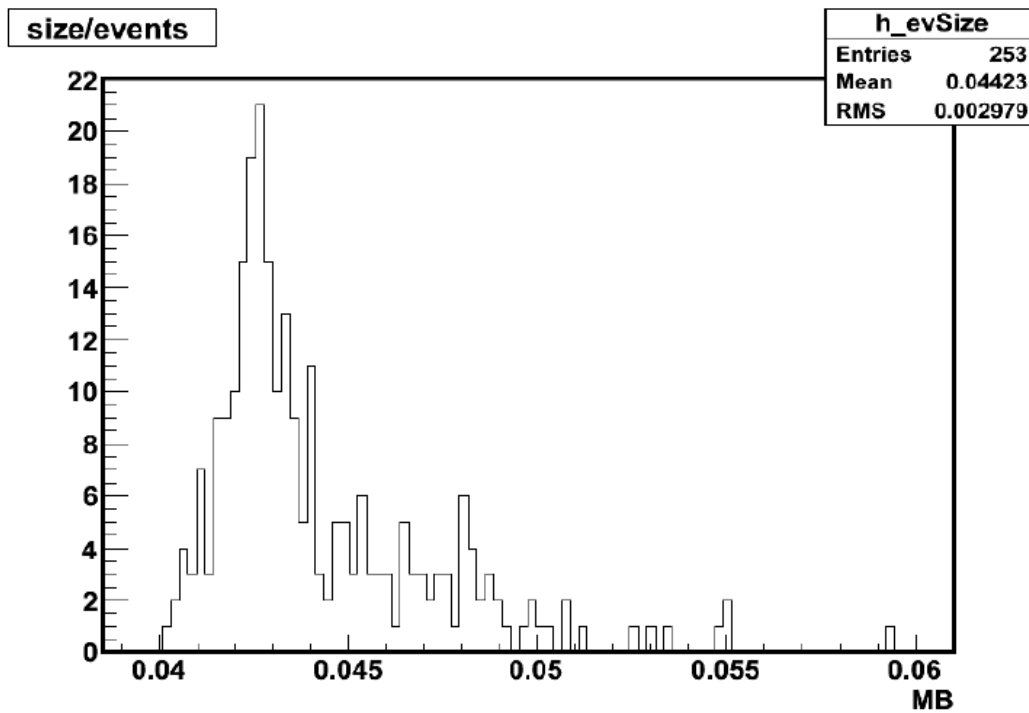


Fig. 4. : read data per event for the test dataset.

2.4. Caching

An interesting effect on the performance analysis is a caching method.

To steer the I/O layer in CMSSW jobs we using I/O adaptors. In this project I have used **TFileAdaptor** – the interface between CMSSW and ROOT. To run the jobs for various type of caching it is necessary to set a value for *cacheHint* parameter. This can be done in configuration file (*patTuple_standard_cfg.py*), by adding the following code:

```
[...]
process.AdaptorConfig = cms.Service("AdaptorConfig",
    cacheHint=cms.untracked.string("lazy-download"),
    # cacheHint=cms.untracked.string("storage-only"),
    readHint=cms.untracked.string("auto-detect") )
[...]
```

Tests were conducted for the following values of *cacheHint*:

- **default (application-only)** - ROOT does the caching.
- **Lazy-download** -Remote files will be downloaded to a local shadow file in 128MB segments. ROOT reads from this local file.
- **Storage-only** - ROOT drives the caching using a prefetch list, but will not allocate a cache of its own.

The *readHint* indicates how I/O reads should be performed.

(**auto-detect**: default and requests optimum read-ahead buffering given the other I/O choices.)

3 Results

Below are the result obtained for the analysis done on the NAF.

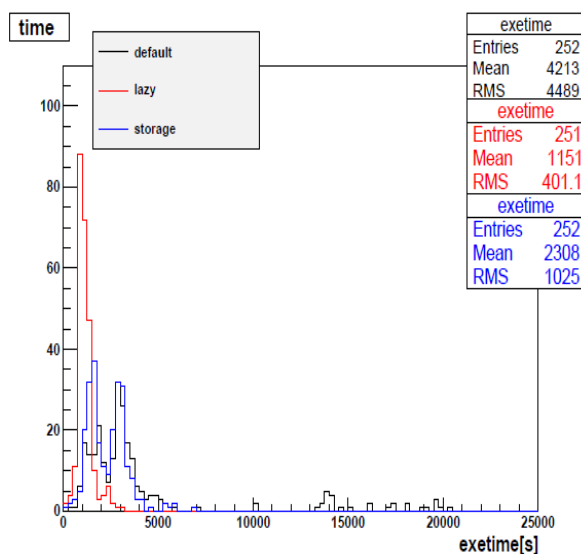


Fig. 5. : Runtime of jobs.

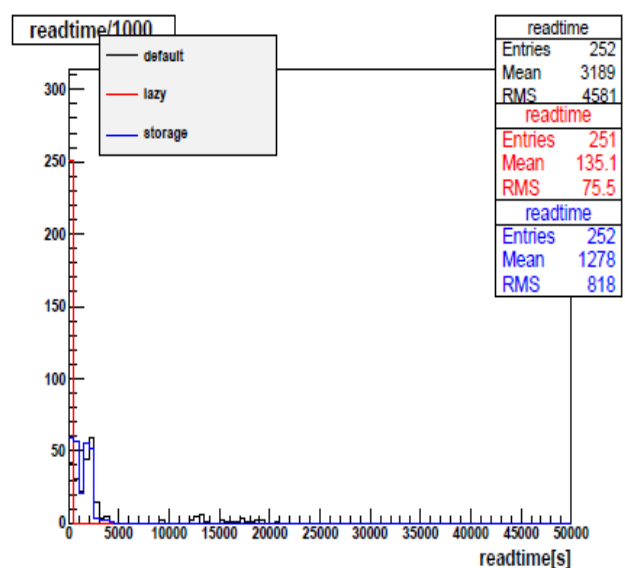


Fig. 6. : Readtime.

Fig. 5 and Fig. 6 show the amount of time used by the individual jobs in the whole analysis process and reading from inputs. We can easily notice that lazy-download setting requires the shortest time to analyze jobs; mean time for one job is 1151 sec (about 20 minutes). Also we want that reading part was as short as possible – for lazy-download setting we get the best performance (mean about 135,1 sec, which corresponds only 10% of the whole analysis time). Default setting is characterized by the worst performance – more than hour analysis of which about 75% is spent for reading data.

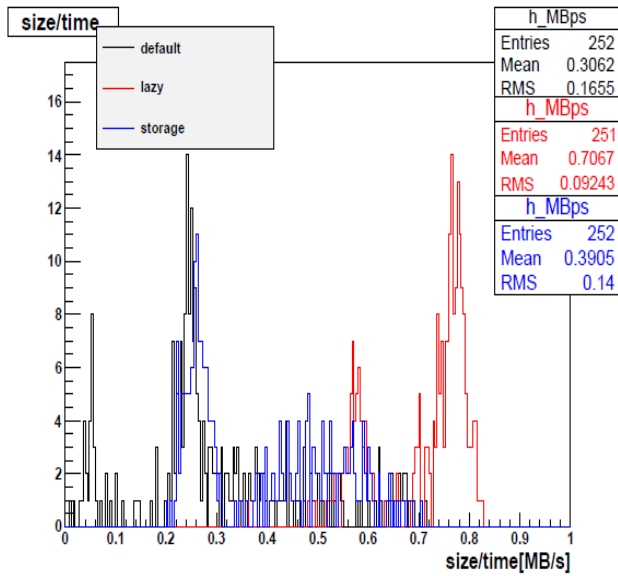


Fig. 7. : transfer (MBps).

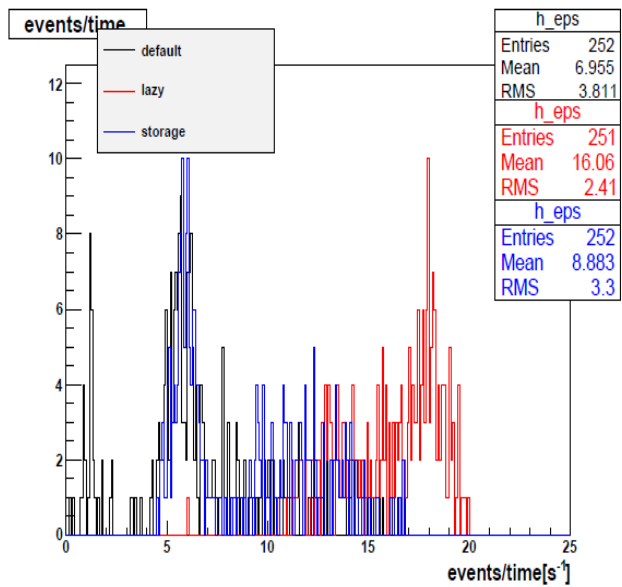


Fig. 8. : transfer (eps).

Histograms on Fig. 7 and Fig. 8 show data transfer rate to the NAF. The best performance is obtained for lazy-download setting; about 0,7 MB per second which is corresponds to 16 read events per second.

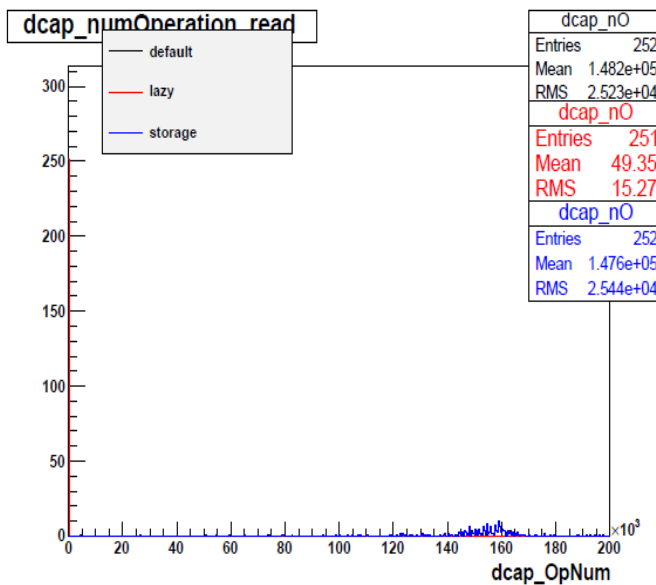


Fig. 9. : number of dcap operations.

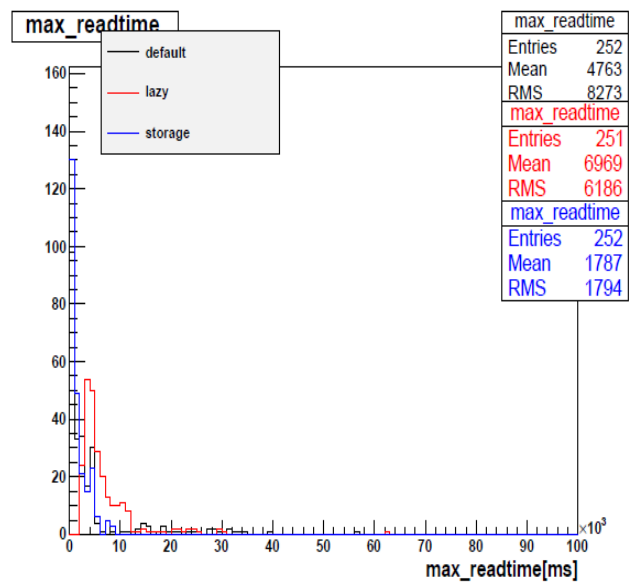


Fig. 10. : maximum readtime.

On *Fig. 9* we can easily see, that the number of dcap operations for lazy-download setting is the smallest (almost three orders of magnitude difference). This is directly related to the largest maximum readtime (*Fig. 10*) for this setting; in this case ROOT reads only 128-MB's segments of local shadow file (these big parts are read in relatively long time, but there is only a dozen such operations).

4 Summary

During the Summer Student Programme 2010 in DESY I have made an analysis of the performance for CRAB jobs on the NAF. I tested the behavior of various caching methods. Proved to be the most efficient method *lazy-download*, which download a remote files to a local shadow file and reads only it.

5 Appendix

In this part I attach plots obtained during the analysis performed for test area 10 GE.

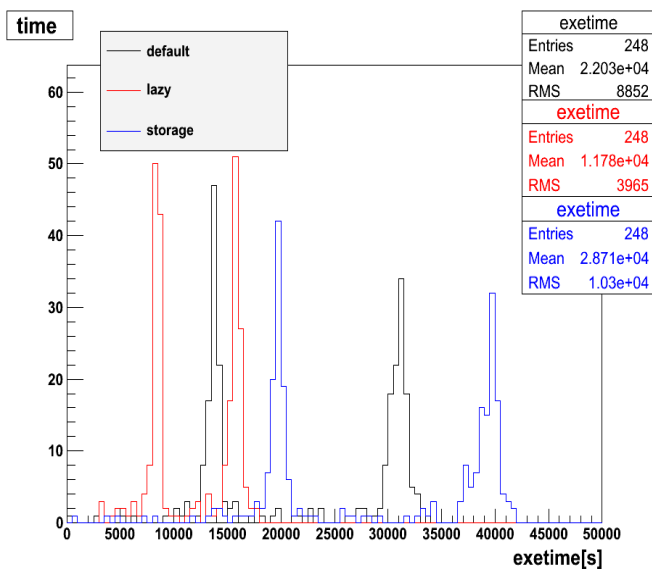


Fig. 11. : 10GE - runtime

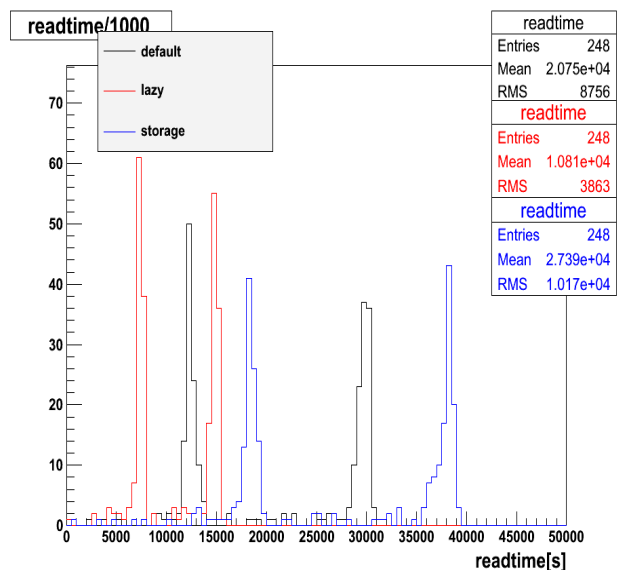


Fig. 12. : 10GE - readtime

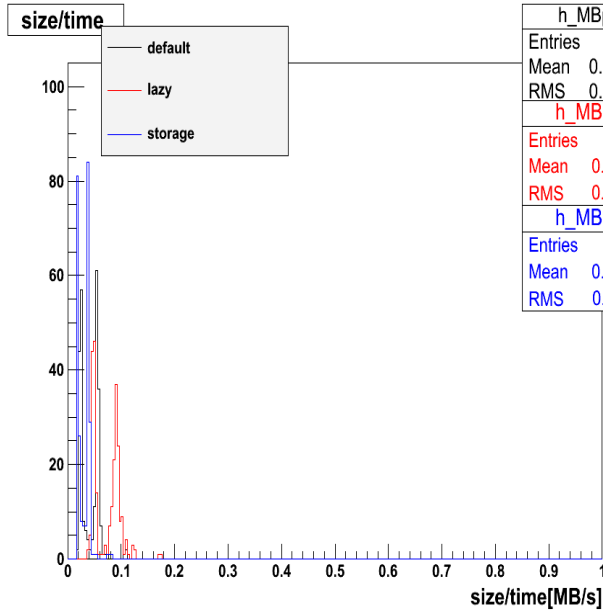


Fig. 13. : 10GE - transfer (MBps)

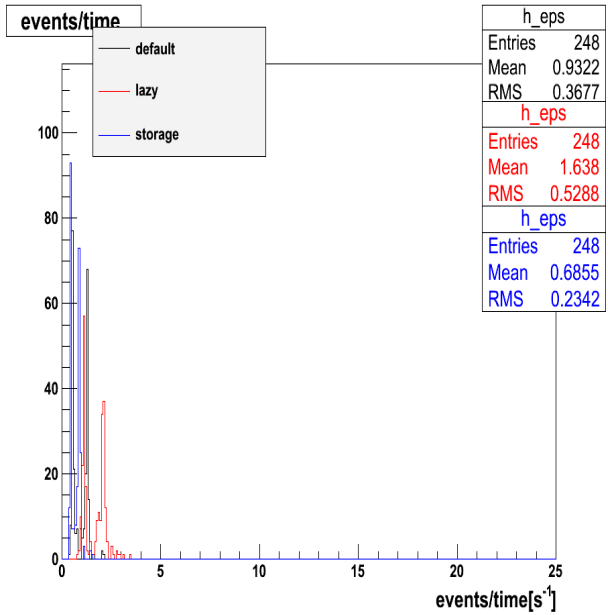


Fig. 14. : 10GE - transfer (eps)

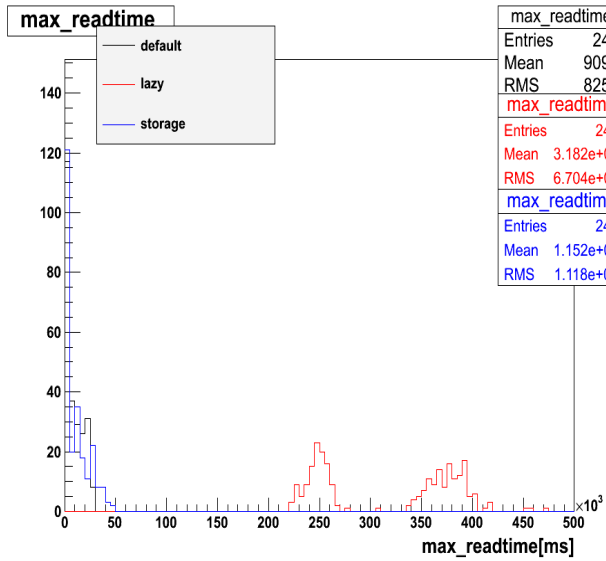


Fig. 15. : 10GE - max readtime

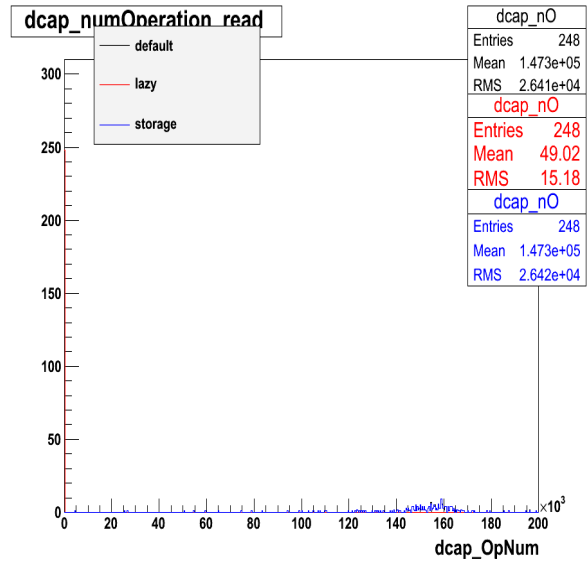


Fig. 16. : 10GE - number of dcap operations

Below I present short description of my scripts.

run.pl: PERL script used to retrieve histograms from ROOT files and merging them.

parsefjr.py: Python script used to parse fjr.xml files and save useful statistics in text files.

draw.C: ROOT macro used to create tree and output histograms.

All_plots.C: C macro used to create comparative plots (for various cache-hint).

Whole of my work was saved in directory: **/afs/naf.desy.de/group/cms/perftests**

References

- [1] **CMS Collaboration:** *CMS, The Computing Project - Technical Design Report, CERN, 2005/07.*
- [2] twiki, *The CMS Offline SW Guide* – <https://twiki.cern.ch/twiki/bin/view/CMS/SWGuide>.