

# PROTOTYPE OF A PARALLEL ANALYSIS SYSTEM FOR CMS USING PROOF

I. González Caballero, D. Cano, R. Marco, Instituto de Física de Cantabria (U. Cantabria – CSIC), Santander, Spain

J. Cuevas, Dpto. de Física, Universidad de Oviedo, Oviedo, Spain

## Abstract

A typical HEP analysis in the LHC experiments involves the processing of data corresponding to several million events, terabytes of information, to be analysed in the last phases. Currently, processing one million events in a single modern workstation takes several hours, thus slowing the analysis cycle. The desirable computing model for a physicist would be closer to a High Performance Computing one where a large number of CPUs are required for short periods (of the order of several minutes). Where CPU farms are available, parallel computing is an obvious solution to this problem. Here we present the tests along this line using a tool for parallel physics analysis in CMS based on the PROOF libraries. Special attention has been paid in the development of this tool to modularity and easiness of usage to enable the possibility of sharing algorithms and simplifying software extensibility while hiding the details of the parallelisation.

The first tests performed using a medium size (90 nodes) cluster of dual processor machines on a typical CMS analysis dataset (corresponding to root files for one million top quark pairs producing fully leptonic final state events distributed uniformly among the computers) show quite promising results on scalability.

## DESCRIPTION OF THE PROBLEM AND USUAL SOLUTIONS

The increasing complexity of the HEP experiments, like CMS [1], and the higher luminosities expected in the new colliders like the LHC, will produce amounts of data in the range of the Petabyte.

The work of an experimental high energy physicist, whether it is devoted to finding a new particle or to measuring some fundamental parameter, consists in selecting some special events from huge amounts of data, both real and simulated. Complex algorithms are implemented to both decide which events might be of interest, and to calculate distributions, efficiency tables, and other types of statistical objects based on the selection. The possible tasks that a typical HEP analysis may need include:

1. Loops inside loops inside loops...
2. Construction of new physics objects (taus, improved electrons, etc) and collections.
3. Simple and complex kinematical cuts

4. Histograms, summaries and various types of mathematical and statistical objects (algorithms, neural networks, likelihoods, etc).

Once these algorithms have been applied on the data and based on the results obtained, new algorithms may be implemented and/or old algorithms may be refined to obtain better results in a continuous cycle.

From the description above, it is easy to infer that it would be highly desirable to produce results as fast as possible in a way that the development of an analysis becomes somewhat interactive. On the other hand it must be noted that the algorithm refining and implementation times may be not small. Thus, the ideal computing model would perform large amounts of data processing, involving fast data access and a great processing power, during short periods of time, followed by close to zero computing activity. In that situation, physicists would spend their time implementing their physics selection code and thinking on ways to improve it instead of waiting for results to come up.

The final stage of the CMS computing model may be done on data files in ROOT [2] format which can be easily produced in the context of the experiment reconstruction framework.

A typical signal sample, like the one corresponding to top-quark pair production with two leptons and two b jets in the final state, may have one million events with a total size of 150 GB in ROOT files. The background samples are usually much bigger. Processing only the signal sample in a single CPU sequentially takes several hours. Processing the background can take days. In such a situation evolving an analysis becomes extremely tedious or even unfeasible. A solution to this problem is to split the samples and use a distributed batch system to process them separately. The results from each sample must be recovered at the end and combined to produce the final plots and numbers. Interactivity is lost in the process and special scripts and macros need to be created to merge the results.

The ideal situation would be such that data can be processed distributed but preserving interactivity. A parallel analysis with sufficient resources may achieve this goal.

## A PARALLEL SYSTEM: PROOF

Parallelising [3] an application is never an easy thing and several issues should be taken into account. In the context of this article parallelisation is done at the level of

events, i.e. each event may be processed in a different CPU, but the algorithms applied to each event are not parallelised themselves.

PROOF [4] is the parallel facility integrated with the ROOT toolkit. It provides a simple framework to process ROOT objects (TTree) in parallel with a small effort. Since data for CMS analysis is expected to be in the form of ROOT trees, PROOF seems the natural solution to parallelise CMS data analysis.

An important aspect not completely related to a parallel system is authentication and certification. It is important to assure a correct access to resources, systems and files. PROOF supports several ways of authentication, including Globus certificates [5], which is a very convenient characteristic in order to profit from the existing clusters with GRID software on them.

In the context of PROOF the computer controlling the parallel processing is called the *master*, while the CPUs processing each of the events are called *slaves*. An efficient communication between the master and the slaves is a must. Within PROOF this tasks is done through a set of specialised daemons.

One of the most important characteristics of a parallel system is the way it handles load balancing. A clever algorithm to distribute the events through the slaves that takes into account network status, data location and CPU performance is a must. PROOF implements a dynamic load balancing mechanism based on the locality of data (events are preferably sent to the slaves where data resides if it is known) and on the performance of each slave.

PROOF also provides an easy mechanism to upload code to the master and the slaves in the form of compressed files with some additional information on how to load the code.

Since data will not be replicated in each slave, rather split over them, it needs to be efficiently accessed remotely. PROOF supports several protocols including the well known RFIO [5] protocol and the more specialised *rootd* [6] protocol.

Finally, merging the results from the slaves might be an annoying task of any parallel system. PROOF takes care of merging objects from the slaves provided they inherit from the ROOT base class (TObject) and implement the right method. This is the case for all main ROOT objects like histograms.

## CMS PROOF

### Tool Objectives

We have developed a tool based on the PROOF libraries. Being a facility implemented by the ROOT development team and fully integrated with it, PROOF seemed like a natural solution to parallelise the analysis of CMS data. The objectives pursued were:

1. PROOF details should be hidden as much as possible so that the physicist can concentrate on the analysis development. Therefore all the operations related to PROOF should be automated and invisible for the user.

2. Migrating code from current analysis applications based on the recommended CMS libraries should be easy with as little modifications required as possible. This means that the tool should integrate those libraries and provide the same level of functionality.
3. Load balancing should be handled and optimised automatically by PROOF. Neither the physicist, nor computer managers should take care of configuring it.
4. A modular approach should be supported facilitating the possibility of code share between physicists and providing a simple mechanism to activate or deactivate different parts of the analysis.
5. Advantage should be taken of the existing GRID local infrastructures. There are CPU clusters ready to use with mechanisms like authentication and certification already in place.

### Implementation

The tool was built around very few classes and some scripts.

The modular approach was achieved with the concept of *Analysis Module*. An analysis module encloses a set of related algorithms to process the data. For example a module may hold the code to identify isolated leptons and to produce the histograms on their kinematics. Then, another module may use this information to select Higgs boson events. A given module can be easily shared among different analysis groups. All analysis modules inherit from a pure virtual base class. Two methods need to be implemented by any analysis module: one with the initialisation code of the class data members that will be called once at the beginning of the run, and another with the actual analysis algorithms that will be called for each event. Some other hooks are provided and the set of possible options will be increased in the future.

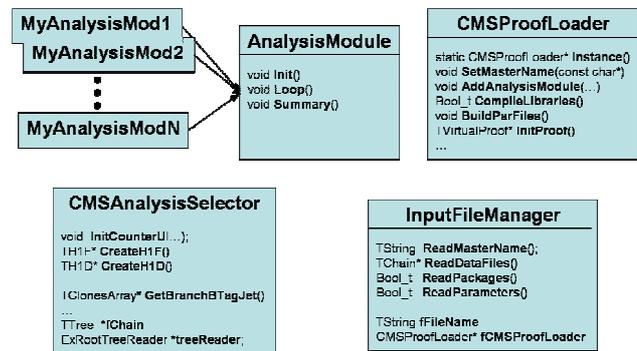


Figure 1: Main classes implemented in the tool with some of the relevant methods specified

A set of related algorithms and classes that do not apply specifically to each event, but that may be used by any other module is called a *Package*. For example, since ROOT does not implement a simple counter class that may be used inside PROOF (normal integers coming from each slave would not be added in the master since

they do not inherit from `TObject`) we implemented it as a package and included it with the tool.

We grouped all the operations related to PROOF in a single class. This singleton class takes care of tasks such as compiling, packaging and uploading the analysis modules to the slaves.

The fact that we did not want the user to take care of load balancing lead us directly to build a class specialising the ROOT `TSelector` class. Additionally this option provided a mechanism to automatically merge the results from the slaves. This class is a kind of glue between PROOF, the data and the analysis modules. It was, therefore, the natural place to integrate the CMS functionality. If histograms and counters are initialised through the methods provided by this class they are automatically registered to PROOF. This is a necessary step for them to be recovered and merged at the end of the session.

Finally, a manager class to handle input from a settings file with a special format was also implemented. Among the things that can be specified in the input file are the master name (mandatory), the location of data files and which analysis modules and packages should be used.

### *User point of view*

Of course a user does not need to know or interplay with all the details of the tool. A typical physicist would only see the input file and the modules he implements. In order to further simplify his life and since all analysis modules have the same shape a special script to generate the skeleton code and directory structure of a module is provided. Therefore a user only needs to run this script, edit the corresponding files that are created and add the module into the input file. Then, from the root session a ROOT macro that is included with the tool may be run. At the end the histograms, counters and other objects specified in the module will be ready in the root session for manipulation (drawing, writing to a file, printing ...).

## PERFORMANCE STUDIES

To test PROOF scalability we have used a cluster of 90 nodes installed at IFCA, Spanish Tier-2 for CMS. Each node consisting on a IBM xSeries 336 with 2 Xeon 3.2 GHz processors, 2GB of memory and 2 SATA Hard disks (80 + 400 GB). The nodes were connected through a 192 ports Gigabit stack switch.

120 GB of data (about 800,000 events) corresponding to top pair production samples with two leptons and two b quarks in the final state were manually distributed through the slaves so that each node would have a single 1.5 GB file with approximately 10,000 events. All data was served to the different slaves via the *rootd* protocol.

We installed PROOF and the CMS software in 80 of the nodes and set it so that only one slave would be sitting in each node.

We run the code corresponding to the study and selection of the events in the sample mentioned above were one of the leptons is a tau decaying hadronically. More than 20 analysis modules (40K lines of code) were implemented, including tau reconstruction, selection of isolated electrons and muons, comparisons between generated particles and reconstructed tracks, etc. About 200 histograms and a verbose summary are produced in the end.

For a better understanding we separated the initialisation and event processing times. Some of the results obtained may be seen in table 1. It is worth noting that while processing the whole sample in a single CPU took more than 4 hours in total, using the full 80 slaves needs less than 8 minutes.

Table 1: Variation of the Initialisation, event processing and total time (in seconds) taken by the analysis described in the text with respect to the number of slaves used

# slaves	Init	Event	Total
1	205	14872	15078
10	202	1412	1615
20	203	713	917
40	205	393	598
60	210	272	483
80	208	235	444

The initialisation time was found to be not small (more than 3 minutes) but surprisingly independent on the number of slaves. A split of initialisation time into the different actions needed is shown below:

- Authentication: It is done on all the slaves a master sees no matter if they are used or not. Some improvement can be achieved by not authentication unless a given slave is to be used.
- Setting remote environment: It was found to be negligible
- Uploading and compiling packages and modules: Being done in parallel this part does not depend on the number of slaves. It is done only for new code. For small modifications it was found to be very small.
- Data chain initialisation: It is by far the most time costing action when long chains are used and distributed.

The event processing time scaled with the number of slaves reasonably well. Figure 2 shows the speed-up quantity versus the number of slaves used. We observed a scaling that did not divert more that 20% from the ideal case extrapolated from the figures for one slave.

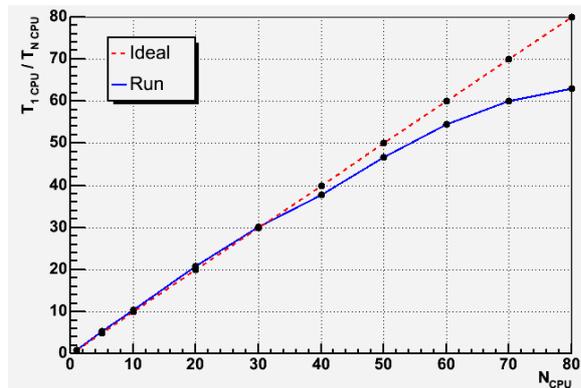


Figure 2: Speed-up plot (event processing). The red dotted line shows the ideal extrapolation from one CPU.

## EXPERIENCE

As already mentioned one of the objectives in the development of this toolkit was to facilitate analysis code migration from the old situation to the new one. Our experience showed that with some simple instructions it was done in a few ours of, basically, copy-paste activity. Some rearrangement of the code had to be done and a few extra lines needed to be written. After this operation, analysis modules where shared between different developers, thus providing consistency to the different analysis and robustness to the code.

Thanks to the fact that we had the possibility to produce a few histograms and other statistical objects in a few minutes, we were able to develop a full analysis very fast, using our time to think on modifications to the analysis and coding the related code.

However there areas were there is still things that can be improved for the toolkit. Debugging code in a parallel or distributed system is never an easy task. Wrong things happening at the remote slaves need to be communicated back to the user session. Changes in the way PROOF itself and our toolkit to handle error messages are needed.

PROOF provides the possibility of visualising the histograms as they are being filled in the remote node. This is certainly very convenient to identify mistakes as early as possible and will be soon implemented in the toolkit.

A delicate task we have found is the direct relation between performance and a correct distribution of the data. The best performance is achieved when data is equally distributed in hard drives of each of the nodes. While this currently needs to be done manually it is expected that PROOF provides an efficient mechanism to handle data to the working nodes. At the same time the possibility of specifying datasets by a logical name (ex. tbar, QCD ...) would be very useful for the users.

From the computer expert point of view PROOF is still difficult to install and tune, being the incompleteness of the documentation one of the reasons. Some problems where also observed like high sensitivity to time synchronisation and instabilities in the two-slaves-per-

node configuration. This situation is rapidly improving and we expect, that together with the new functionality being added to it, PROOF will allow us to also enhance our toolkit.

## CONCLUSIONS

We wanted to implement a tool to quickly, easily and interactively develop a HEP analysis in the context of CMS that would let physicist concentrate on physics, share their code, and that would not force them to rewrite the already existing code. At the same time we wanted to fully exploit our local CPU farms

We built a light tool which profits from the functionality in the existing CMS libraries and the PROOF facility, that fills most of our requirements and that brought “interactivity” back into the analysis cycle. Code was easily and quickly migrated enable code sharing between developers.

In the mean time we gain a lot of experience on using PROOF and developing tools for it, that will allow new functionalities into our toolkit as they come out from PROOF. We also expect this experience will ease the integration of parallelisation in the new CMS event data model and framework.

## ACKNOWLEDGEMENTS

We would like to thank all the IFCA groups for providing the hardware facilities and understanding the exclusive CPU requests we needed for the performance tests.

PROOF development team, and in particular G. Ganis has been extremely collaborative. We would like to thank them for their patience in solving our problems and quickly fixing the few bugs we were able to spot.

## REFERENCES

- [1] CMS Collaboration, “CMS Technical Proposal”, CERN/LHCC 94-38, LHCC/P1, 15 December 1994
- [2] Rene Brun and Fons Rademakers, “ROOT - An Object Oriented Data Analysis Framework”, Proceedings AIHENP'96 Workshop, Lausanne, Sep. 1996  
Nucl. Inst. & Meth. in Phys. Res. A 389 (1997) 81-86.
- [3] “PROOF - The Parallel ROOT Facility”, in preparation (<http://root.cern.ch>)
- [4] See for example the book "Sourcebook of Parallel Computing" by Jack Dongarra, Ian Foster, Geoffrey Fox, William Gropp, Ken Kennedy, Linda Torczon and Andy White.. Morgan Kaufman, 2003. ISBN: 1558608710.
- [5] I. Foster, C. Kesselman, “A Metacomputing Infrastructure Toolkit”, Intl J. Supercomputer Applications, 11(2):115-128, 1997.  
See also: <http://www.globus.org>