A PROOF Analysis Framework

I. González Caballero[1], A. Rodríguez Marrero[2], E. Fernández del Castillo[2] and A. Cuesta Noriega[1]

[1]Physics Department, Universidad de Oviedo, Oviedo, Spain
[2]Instituto de Física de Cantabria, Universidad de Cantabria CSIC, Santander, Spain

E-mail: Isidro.Gonzalez.Caballero@cern.ch

Abstract. The analysis of the complex LHC data usually follows a standard path that aims at minimizing not only the amount of data but also the number of observables used. After a number of steps of slimming and skimming the data, the remaining few terabytes of ROOT files hold a selection of the events and a flat structure for the variables needed that can be more easily inspected and traversed in the final stages of the analysis. PROOF arises at this point as an efficient mechanism to distribute the analysis load by taking advantage of all the cores in modern CPUs through PROOF Lite, or by using PROOF Cluster or PROOF on Demand tools to build dynamic PROOF cluster on computing facilities with spare CPUs. However using PROOF at the level required for a serious analysis introduces some difficulties that may scare new adopters. We have developed the PROOF Analysis Framework (PAF) to facilitate the development of new analysis by uniformly exposing the PROOF related configurations across technologies and by taking care of the routine tasks as much as possible. We describe the details of the PAF implementation as well as how we succeeded in engaging a group of CMS physicists to use PAF as their daily analysis framework.

1. Motivation

The increasing complexity of HEP detectors and the high luminosities achieved in colliders like the LHC produce amounts of data in the range of several petabytes. The common practice to analyze this data follows a path that aims at increasing the value of the contents, and at the same time at reducing the size of the data that needs to be processed. During raw data reconstruction (applied on both real data and MC generated data) complex objects are built by combining a constellation of small measures. By filtering the real data according to some trigger or combination of triggers, and by splitting the MC samples according to the process simulated each analysis can use only those data sets that are relevant to them. A selection of reconstructed objects restricted to those important for analysis produces several hundreds of datasets in the so called Analysis Object Data (AOD). The size per event of AOD is normally around 10% of that of the raw data.

A typical analysis in an LHC experiment, whether it aims at finding a new particle or at measuring some fundamental parameter of the Standard Model, needs to process several datasets adding up to a total size of several hundreds of terabytes. The complexity of the analysis developed in the LHC imply all kinds of algorithms and the understanding of the objects used (electrons, muons, jets, missing traverse energy, etc.). Access to almost all the information stored in the AOD files is needed in many cases. However, the final stages frequently consist on

filtering the interesting events by applying a more or less complex set of selection criteria over a reduced set of usually high level event objects.

At this point more or less 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 to include are:

- Several levels of loops to traverse collections of objects
- Construction of new physics objects (taus, improved electrons and muons, etc) and collections.
- Simple and complex kinematical cuts
- Histograms, summaries and various types of mathematical and statistical objects (algorithms, neural networks, likelihoods, etc).
- Various scale factors, event re-weighting, etc..

Algorithms are refined and new ones are implemented to obtain better results in a continuous cycle during the analysis development.

The use of Grid technologies allows the HEP community to access and process the big volumes of data in a reasonable time at the cost of excluding any kind of interactivity during data processing and by adding some complexity to the computing operations. On the other hand, it is easy to infer that, during the last steps of an analysis, it would be highly desirable to be able to produce and inspect results as fast as possible. It should be noted that the time spent in refining old algorithm and implementing new ones may not be 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. Figure 1 shows an schematic view of the different CMS data formats as it evolves towards data used in analysis. The tier level which is used to get each of the steps is also shown.

Small clusters of computers (like Tier-2 and Tier-3 centres) with spare computing cycles are common in many research institutes. Modern computers with 8-12 cores can be acquired at reasonable prices nowadays. Many analysis groups apply an strategy consisting on reducing the event size by both storing only the information needed for the analysis and by flattening the event structure complexity, getting the total size of the data needed to be analyzed down to a few terabytes of files containing a ROOT [1] flat tree. This approach has the advantage of simplifying the analysis code by suppressing any dependency on specific collaboration framework software, with the exception of ROOT itself, leading to an easier usage of the computing infrastructures mentioned before. One could at this level split the samples and use a distributed batch system to process the various bits separately. The results from each sample must be recovered at the end and combined to produce the final plots and numbers. Interactivity is again lost in the process and special scripts and macros need to be created to merge the results.

We believe that a better and more integrated option is provided by the Parallel ROOT Facility (PROOF) [2].

2. The Parallel ROOT Facility, PROOF

Parallelizing [3] an application is never an easy task and several issues should be taken into account. In the context of this article parallelism is achieved at the level of events, i.e. the computing load is distributed among the available computing units by having them perform the whole set of operations over a given subset of the events. The algorithms applied to each event are not parallel themselves.

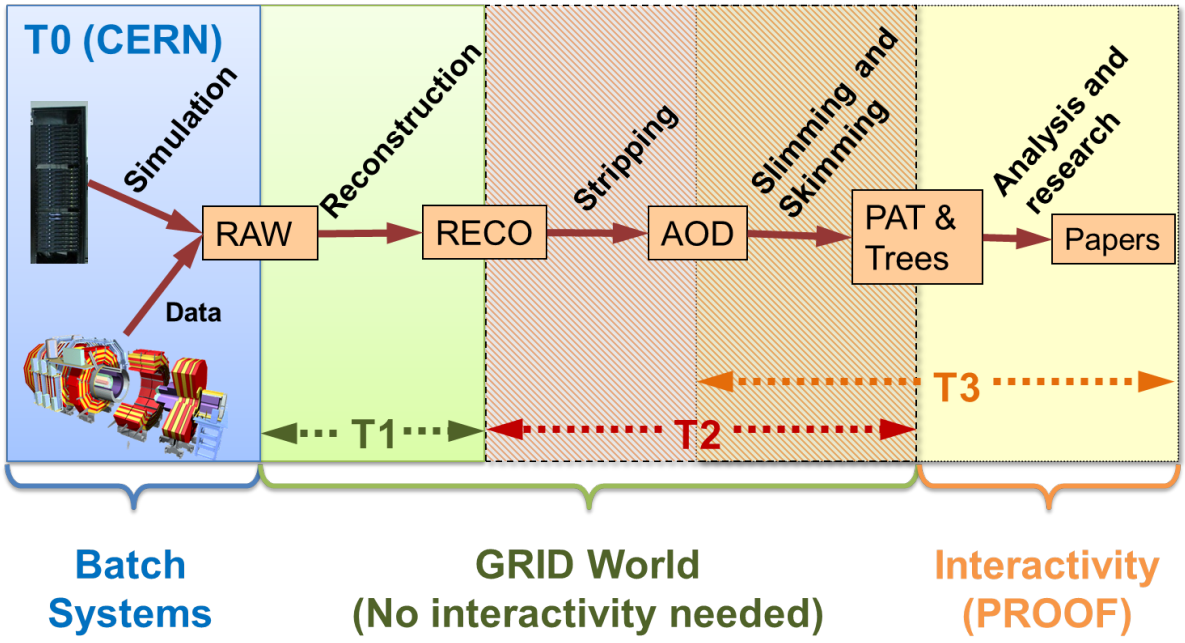

Figure 1. Evolution of the CMS data formats toward data used in analysis. Both size per event and size per sample is reduced as we move left. Reducing the size of the data is important to speed its processing. It is also worth noticing that as we move left data needs to be process more times.

PROOF is the parallel facility integrated in the ROOT toolkit. It provides a simple framework to traverse ROOT main collections (TTree) in parallel with a small effort. PROOF has been designed with the following goals in mind:

- Transparency: The analysis code should need as little modifications as possible to be run on PROOF with respect to being executed sequentially.
- Scalability: the basic architecture should not put any implicit limitations on the number of computers that can be used in parallel.
- Adaptability: the system should be able to adapt itself to variations in the remote environment (changing load on the cluster nodes, network interruptions, etc.).

PROOF is therefore a natural solution to speed and improve the last stages of HEP data analysis.

One of the most important characteristics of a parallel system is the way it handles load balancing. A clever algorithm to distribute the events over the worker nodes that takes into account network status, data location and CPU performance is a requirement, specially when dealing with heterogeneous clusters. PROOF implements a load balancing mechanism that dynamically distributes the load based on the performance of each computing unit.

Since data will not be replicated in each worker node, rather split over them, it needs to be efficiently accessed remotely. ROOT supports several protocols such as RFIO, rootd, http and posix compliant file system. That means that most of the storage systems used in HEP can be used from PROOF. A wrong or poor selection of the storage technology can be a bottleneck in getting the data to the worker nodes giving a bad performance for a PROOF based analysis. Special attention should be paid to this aspect when designing the analysis strategy.

Finally, merging the results from the worker nodes might be an annoying task of any distributed system. PROOF takes care of merging objects from the worker nodes 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 and collections.

3. PROOF Modes

As already hinted before, PROOF aims at both being able to benefit from the many-core modern architecture of a typical desktop or laptop, and the availability of computer farms (ex. Tier-2s, Tier-3s, central analysis facilities,...).

In the context of PROOF the user starts a session on a PROOF *client* by connecting to a PROOF *master*: a computer controlling the parallel processing. This machine is aware and connected to a set of computing units called *workers*. While no special installation is needed in the client apart from the ROOT toolkit, both in the master and in the workers special daemons need to be deployed and configured. While efforts have been done to provide enough documentation and tutorials to correctly install and deployed a PROOF farm like that, it is still a complicated task that needs quite some level of expertise and understanding of the system due to the high level of flexibility and variability of the resources that might be available. Furthermore, superuser access is normally required for this.

However most of the computing facilities at HEP institutes already implement batch systems that can be exploited in order to build dynamic PROOF clusters. In this way resources need not be dedicated to PROOF nor specifically configured. If special queues with fast access times are available, the computing resources needed can be allocated and freed as they are required. This is expected to happen not so often for any particular user. There are several tools that try to facilitate the interaction of PROOF with the batch system by hiding most of the complexities below:

- PROOF Cluster [4]: Developed in the context of the Tier-2 for CMS [5] at IFCA, it supports SGE and PBS batch systems. It has been designedd with simplicity in mind. Setting the PROOF Cluster environment can be done pretty quickly. It requires the installation of a configured master to deal with the client-workers communication which can be optimized and tuned to the specific characteristics of the cluster in which it is deployed (storage model, authentication,...).
- PROOF on Demand (PoD) [1]: Developed at GSI, supports a wide variety of batch systems as well as password-less ssh accessible clusters. The master is also created dynamically and therefore no administrative privileges are required to deploy it.

Given the emergence of cloud computing technologies and the availability of cloud enabled computer farms efforts have been taken to integrated PROOF with them. The PROOF Cloud [6] toolkit aims at providing an efficient way to bridge the two technologies giving the users access to a larger and more flexible number of calculation resources.

Finally, the PROOF team themselves provide a out-of-the-box solution to take full advantage of the additional cores available in nowadays desktops and laptops called PROOF-Lite. Since no special installation is required it provides a cheap and easy way to both analyze not so big data samples and to start operating with PROOF when a many-core machine is at hand.

4. PROOF Analysis Framework, PAF

4.1. Goals

From the point of view of the users, using PROOF at the level required for a serious analysis introduces some difficulties that may scare new adopters. For this reason we have developed the PROOF Analysis Framework (PAF) to facilitate the development of new analysis by uniformly

[1] See http://pod.gsi.de

exposing the PROOF related configurations across technologies and by taking care of the routine tasks as much as possible. The objectives pursued in the design of PAF are:

(i) 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.

(ii) Migrating traditional analysis code should be easy with as little modifications required as possible.

(iii) The user should be able to run exactly the same analysis code using any of the PROOF modes mentioned in the previous section.

(iv) A sequential mode where none of the PROOF specific functionality is present should be available. This is very useful for debugging code since one can very easily find if problems come from the user code or from the PROOF implementation, or, even from the computer farm configuration.

(v) Advantage should be taken of the existing computing local infrastructures with minimal needs for special configurations.

The global goal is that physicist concentrate on the analysis needing as little knowledge of the computing framework they are using as possible.

PAF has been built around a very few classes and some scripts. The details of the implementation of the analysis lay on a specialized ROOT selector class, while all the interaction with PROOF happens through a manager object.

4.2. Analysis Code

In order to profit from the full functionality inside PROOF the analysis code has to be packaged on a class inheriting from `TSelector`[2]. This class needs to be aware of the whole tree structure and, therefore, needs to be in sync with changes on the structure of those trees. Even if PROOF provides some tools to generate the skeleton of that class automatically, moving code around whenever changes happen on the data structure, is painful and prone to errors. Through a set of inheritance levels and by using the module uploading capabilities within PROOF, we have gone one step forward avoiding the manual recreation of the branches every time the exact content of the data files changes. We are able of produce, compile and load that information dynamically and efficiently. No CPU cycles are wasted. User code stays packed in the same place and unaltered. Users can, therefore, concentrate the efforts on coding the actual selection and reconstruction algorithms for the analysis.

A set of extra tools specialized for its usage in PROOF are also provided: counters, input parameters, and generic collections. For example, since ROOT does not implement a simple counter class that may be used inside PROOF (normal integers coming from each worker would not be added in the master) we implemented it as a package and included it with the tool. Another class allows the user to add input parameters of different types to the analysis that can be set outside of the selector class, adding flexibility to the analysis code.

The handling of commonly used analysis objects (histograms, counters, collections) is optimized through dedicated methods in order to take care of repetitive calls. If histograms, trees 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 a PROOF session.

With this design a user would need to inherit from our specialized class. Then there are four virtual methods that can be implemented which take care of:

[2] See, for example, http://root.cern.ch/drupal/content/developing-tselector

(i) Parameter initialization: Finding the configuration parameters. Its implementation is not mandatory. It is one of the first methods called.

(ii) Initialization of analysis result objects: In this method one typically initialises histograms, counters or trees that will be filled at each event. It is called just before getting into the event loop.

(iii) Event processing: Here is where the user should put the code to select the interesting events and to fill the analysis result objects with the correct values according to the analysis strategy adopted. Access to all the event branches is available with no special action needed.

(iv) Final operations: After merging the objects from the different workers one may want to perform some additional operations (output some results, additional calculations,...). This is the place where the corresponding code goes. Most of the time nothing needs to be done.

4.3. PROOF Configuration

We have grouped all the configuration possibilities in a global object belonging to a single class (`PAFOptions`). A couple of functions take care of initializing PROOF and running the analysis based on the options selected. This is normally started and set in a ROOT macro. The result objects created in the PROOF session are automatically stored in a root file for later inspection. The users may also select some of the histograms to be interactively plotted as they are filled during a PROOF session. This may be very useful in early spotting mistakes in the analysis code. A non-exhaustive list of options follows:

- PROOF Mode: This tells PAF which of the supported PROOF mode should be used to run the analysis. The current possibilities are sequential (no PROOF), PROOF-Lite, PROOF Cluster, PoD and PROOF Cloud. Some specific values for some of the modes (master name or port number for PROOF Cluster) can also be set.
- Number of slots: The number of computing units PROOF should try to match. If not enough computing units are found within a reasonable time PAF will run on the slots available at that moment.
- Name of the specialized selector class implemented by the user.
- Data files to be processed. The first data file is used to find the tree structure and to generate the intermediate selector class.
- Name of the output file with the analysis result objects constructed in the analysis.
- Input parameters: An instance of the `InputParameters` class developed in the context of PAF where specific analysis configuration values can be passed (ex. full luminosity, cross section, etc.).
- Dynamic histograms: Name of some of the histograms being filled in the analysis that the user wants to see as it is being filled.
- Additional analysis packages: PROOF provides a mechanism to upload code modules packaged in a specific way that can be afterwards used by the specialized selector in the workers. PAF takes care of the whole task of correctly packaging this modules by generating the necessary extra files that PROOF requires.

4.4. Running PAF

Of course a user does not need to know or interplay with all the details of the tool. A typical physicist would start by building the analysis code within the specialised selector class and then continue modifying the configuration macro to adapt it to the particular environment in which the analysis is run. Example files are provided for all these files. The macro file is rarely modified while the analysis algorithms evolve. From a ROOT session the macro is invoked and

the analysis is executed. At the end of the processing the output file is reopened automatically so all the histograms, counters and other objects specified in the module are available for quick inspection and manipulation (drawing, printing,...). Figure 2 shows a PAF session using 10 workers cores in a dynamic PROOF cluster.

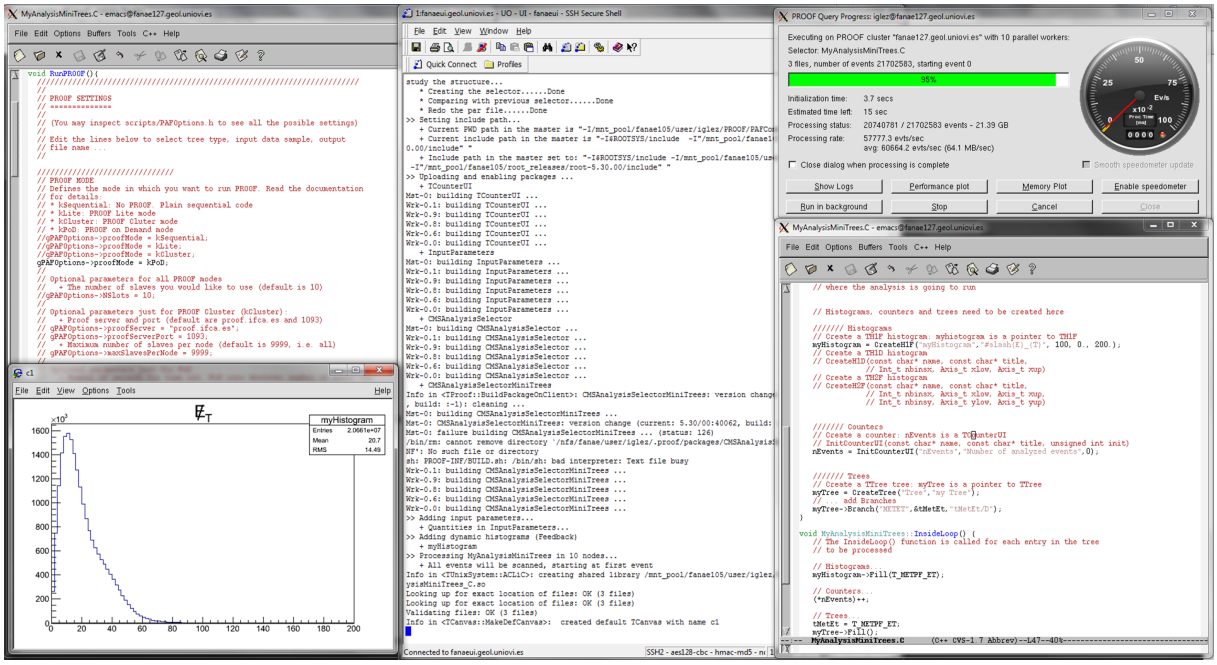

Figure 2. An example of a PAF session using 10 cores in a dynamic PROOF cluster. A histogram is shown bottom right window with the event transverse missing energy distribution that is updated as events are processed. The grey background window on the top is automatically launched by any PROOF session and provides updated information on several performance parameters as well as access to the logs. A snippet of the main macro code (top left) and the specialised selector code (bottom right) can also be seen.

5. Experience and future enhancements

PAF has been developed in the HEP groups of the University of Oviedo and IFCA to improve the development of analysis for CMS data.

The migration from the old framework to the new one based on PAF, since it mainly implied a reshuffling of the existing code, happened within a few days. The gains in speed achieved both in local desktops connected to an efficient storage system, and by using the Tier-2 infrastructure at IFCA and the Tier-3 infrastructure in Oviedo, convinced the users of the benefits of this approach (see figure 3). By decoupling the analysis from the heavy CMS software framework (CMSSW) we also guarantee that the requirements in terms of memory of the workers stays stable well below 500 MB on a normal PROOF session. This is far from the usual requirements of a full CMSSW session typically well above a gigabyte.

PAF is currently used continuously at IFCA and U. Oviedo. It is the main framework for the final stages of the analysis in which both groups are involved. The collaboration results in which both groups have responsibilities have been ultimately produced using PAF. Those results include the study of $t\bar{t}$ and W^+W^- production as well as Higgs boson ($H \to W^+W^- \to \ell^+\ell^-$ channel) and SUSY searches.

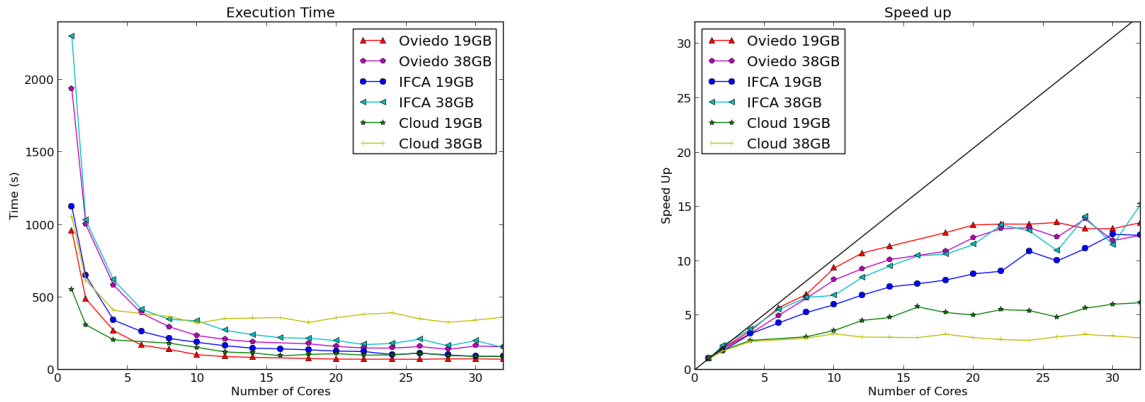

Figure 3. Time spent (left) and speedup factor (right) using PAF over a different number of workers. Two samples of $t\bar{t}$ MC events weighting 19 GB and a 38 GB were used. Three different PROOF modes are shown: PROOF Cluster (IFCA), PROOF Cloud (Cloud) and PoD (Oviedo). No significant gain is found above 20 nodes. This is thought to be due to the simplicity of the test and the high but limited bandwidth of the storage systems (GPFS at IFCA and HDFS at Oviedo)

New members of both groups typically start reproducing results in a couple of days and are able to evolve the analysis code within a week. Of course, this is possible also due to the documentation provided to our members and the support provided by the developers.

We are exploring the possibility of further enhancing a modular approach to the analysis by providing a mechanism that would iterate over a collection of PROOF packages that follow a well established structure. 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 like that can be easily shared among different analysis groups. While this can be achieved with the current functionality in place in PAF (and it is actually done) the proposed mechanism would make sharing modules and objects in modules easier. The analysis studies would become more flexible and a repository of modules could be built.

Since PAF was first implemented, PROOF has evolved providing additional functionalities and enhancements that we have not yet integrated. Those include an efficient use of temporary storage for large objects, like for example large output trees, that can reduce the memory consumption and increase the object merging speed. We intend to follow the latest developments and apply them to our framework.

We have tuned PAF to understand our CMS flat trees, but it could be very easily generalised to any other kind of tree based data. Other experiments approaching analysis the same way could use PAF.

6. Conclusions

We have developed a framework based on PROOF to distribute the computing load and benefit from the many-core modern architectures, as well as the existing frameworks in HEP institutes, that addresses in a easy way some of the problems found at the last stages of typical HEP analysis. PAF has proven to be a tool that is easy to adopt and that can highly increase the speed at which analysis algorithms are developed and evolved by achieving fast response times with no extra computing knowledge required, and by bringing interactivity back in the process.

Competitive and official results have been produced in CMS using PAF and new users are able to work with it in a very short time.

References

[1] Brun R and Rademakers F 1997 *Nucl. Inst. & Meth. in Phys. Res.* A **389** 81-86. See also http://root.cern.ch/.

[2] Ballintijn M, Biskup M, Brun R, Canal P, Feichtinger D, Ganis G, Kickinger G, Peters A and Rademakers F 2006 *Nucl. Inst. & Meth. in Phys. Res.* A **559** 13-16.

[3] Dongarra J, Foster I, Fox G, Gropp W, Kennedy K, Torczon L and White A 2003 *"Sourcebook of Parallel Computing"* Morgan Kaufman, 2003. ISBN: 1558608710.

[4] Rodríguez Marrero AY, González Caballero I, Cuesta Noriega A, Marco de Lucas J, Matorras Weinig F 2011 *J. Phys.: Conf. Ser.* **331** 072061

[5] CMS Collaboration 2008 *The CMS experiment at the CERN LHC JINST* **3** S08004

[6] Rodríguez Marrero AY, González Caballero I, Cuesta Noriega A, López García A, Marco de Lucas J, Matorras Weinig F *Integrating Proof Analysis in Cloud and Batch Clusters* to be published at the Proc. for CHEP2012