PROOF Analysis Framework (PAF)

J. Delgado Fernández¹, E. Fernández del Castillo², I. González Caballero¹ and A. Rodríguez Marrero² for the CMS Collaboration.

¹Physics Department, Universidad de Oviedo, Oviedo, Spain
²Instituto de Física de Cantabria, Universidad de Cantabria CSIC, Santander, Spain
E-mail: Isidro.Gonzalez.Caballero@cern.ch

Abstract. The PROOF Analysis Framework (PAF) has been designed to improve the ability of the physicist to develop software for the final stages of an analysis where typically simple ROOT Trees are used and where the amount of data used is in the order of several terabytes. It hides the technicalities of dealing with PROOF leaving the scientist to concentrate on the analysis. PAF is capable of using available non specific resources on, for example, local batch systems, remote grid sites or clouds through the integration of other toolkit like PROOF Cluster or PoD. While it has been successfully used on LHC Run-1 data for some key analysis, including the $H \rightarrow WW$ dilepton channel, the higher instantaneous and integrated luminosity together with the increase of the center-of-mass energy foreseen for the LHC Run-2, which will increment the total size of the samples by a factor 6 to 20, will demand PAF to improve its scalability and to reduce the latencies as much as possible. In this paper we address the possible problems of processing such big data volumes with PAF and the solutions implemented to overcome them. We will also show the improvements in order to make PAF more modular and accessible to other communities.

1. Motivation
The huge amount of resources required to analyze the data produced by the LHC requires the development of dedicated tools able to efficiently handle it. The experimental data recorded at the LHC experiments typically follow a set of steps aiming at producing higher level physics objects in a filtered dataset with only the relevant events and observables needed in a particular analysis. In the last phase of the analysis, very often, the remaining data is stored in ROOT [1] files with a tree like structure with still a large number of entries (several millions or tens of millions) corresponding to each of the events selected. The ROOT framework provides all the functionality required to process the data and to produce aggregated information like histograms, summaries or other ROOT files. The Parallel ROOT Facility (PROOF) [2] is an extension of ROOT enabling the execution of analysis workflows over parallels systems like multi-core machines, computer clusters or more complex solutions.

However, the complexities of C++, ROOT and PROOF needed in the development and execution of an analysis produces a steep learning curve that frequently scares new users. Furthermore, the implementation of a new PROOF based analysis often requires some auxiliary functionality that may detach the user attention from the physical details of the analysis itself. A HEP analysis is usually a collaborative task where several people contribute at different levels. Though it is always possible to establish good code sharing mechanisms the current PROOF simple framework based on a monolithic TSelector derived class is far from enforcing it.
The PROOF Analysis Framework (PAF) has been developed with the target of producing a comfortable to use tool, with its design driven by the real needs of HEP analysis users and aiming to solve the problems hinted in previous paragraphs. Users should only focus their attention on the physical problem they are addressing leaving aside all the technical issues and most of the configuration details.

The PAF we are describing here [3] is a continuation of the work presented in [4]. The new PAF is not only a re-design and re-implementation of the previous framework but it has also been extended with new capabilities.

In section 2 we describe the principles that drove the redesign and re-implementation of PAF. Section 3 has the details of the main functionality provided by PAF. The information on the execution environments supported by PAF may be found in section 4. The results of some performance tests done with the new version of PAF is in section 5.

2. Design principles

The new framework has been developed with a strong object orientation using mainly C++. Some aspects have been coded using pure bash shell scripts and templated files. Bearing in mind the easy modification and extension of our proposed solution we have made a healthy abuse of design patterns, specially strategy patterns, designing basic interfaces to almost all the functionalities included in PAF that are later specialized for common situations. This leads to a framework where almost any behavior can be modified or adapted during the analysis execution.

The new PAF has also a more modular organization of the code which is now distributed in a hierarchical and logical structure of directories, putting together the code with the same responsibility.

Although this new version of PAF has been rewritten almost from scratch, we have a good backward compatibility to previous versions. Minimal modifications are needed for users of previous versions. The changes concentrate on the way data branches are accessed and in the job configuration where we preferred to break compatibility in order to simplify further the user code.

PAF has been tested successfully under GNU/Linux with Scientific Linux CERN, Ubuntu and Debian distributions using the GNU Compiler Collection (GCC) compiler as well as under OS X Yosemite (10.10) using the CLang compiler. Though the current release (5.0) has been developed to run under ROOT versions 5.32+, it will soon support also ROOT 6.x.

3. PAF Features

3.1. Lazy loading of branches

Most of the time not all the branches stored in a ROOT file are needed for a given analysis. The previous PAF releases accessed all the branches for each event by automatically including new variables in the main analysis class. This had two drawbacks that have been now addressed: it increased the complexity of the implementation of a modular analysis (see 3.3) and it forced ROOT to read all the information in the file which resulted in a waste of time and resources.

The new PROOF Analysis Framework provides a different way to access the values stored in the branches of ROOT files like if it was a dictionary. In order to get a variable several functions where the branch name is passed as a key are implemented. PAF takes advantage of this design to dynamically inform ROOT of the branches used as they are been accessed at run time so it can optimize the data reading. By default, each analysis starts with no branch marked for loading. When the user tries to get any branch or variable the framework internally checks if it has been selected for loading before. Branches not been accessed before are marked for loading in future events.

This lazy loading of branches introduces a strong boost in performance (see section 5 for more details) speeding the analysis by lowering the input bandwidth used. The reduction in the
amount of data been read the system becomes also more storage friendly.

3.2. Common functions simplified

One of the goals of PROOF Analysis Framework is to help newcomers with limited knowledge of ROOT, PROOF and PAF. By providing an easy to access implementation for common tasks PAF not only cuts the learning curve of inexperienced users, but it also allows expert scientists to become more productive. Specifically, it provides support to create histograms, profiles, ROOT trees and files to process the data.

Providing logging output in a distributed environment is always a difficult task. But at the same time it is very useful to debug and validate the execution of a workflow. We have implemented an easy-to-use and powerful logger widely used in PAF to record the framework run time events. The logger has filtering capabilities based on a level scale (info, debug, warning, error and fatal) that can be tuned by users. The output can be printed in the console or sent to a file. The logger behavior can be easily modified or extended by specializing a class.

PAF also contains a tool to easily inspect the content of the input files. This tool has been designed to be used with ROOT files and it shows the name of all the branches found in a file and the type of the data for each branch. It even provides a code snippet that may be copied directly to the analysis classes. A particular tree can be specified for input files holding multiple trees. Furthermore, it has been enhanced with the possibility to use regular expressions to filter branches based on patterns.

3.3. Modular analysis

One of the biggest evolutions in the new PAF described in this paper is the possibility of creating analysis composed by several analysis modules (selectors). As time goes by, analysis tend to become more and more complex involving several collaborators. At the same time several analysis may share the same algorithms for certain steps (for example lepton selection or missing transverse energy estimation). The modular analysis framework implemented contributes to facilitate the sharing of code among similar analysis, or the distribution of responsibilities in a complex analysis. The class diagram is shown in figure 1.

Furthermore, we believe that by structuring any analysis in several simpler and isolated modules the whole implementation may become more flexible and manageable helping the developers to concentrate in small problems. Another advantage of this approach is that if a modification is needed, changes will be localized and they should not affect the rest of the code.

The communication between selectors is implemented following the strategy developed for finding branch content. Each selector can create a new variable which can later be accessed in any other selector. This technique have shown to be computationally light.

An example of a possible modular analysis is shown in figure 2 where each module may create new variables, needed in the following selectors.

![Diagram](image-url)
3.4. Multiple repositories
This functionality is heavily related to the one described in the subsection above. With the idea of improving the collaboration among different research groups PAF provides a way to define several module repositories. They are defined in a typical shell environment path variable so that modules are automatically searched in the specified locations. The paths are scanned in order and the modules are loaded from the first available location.

3.5. Compilation optimization on heterogeneous clusters
One of the actions PAF automatically does is the compilation of the user code and modules. We have foreseen two main situations: homogeneous and heterogeneous infrastructures.

In homogeneous clusters one can rely on a single compilation and the compiled files and libraries been distributed to the slaves. By compiling the code just once in a single machine we can spot compilation problems at an early stage and avoid the creation of a PROOF session if it fails. It may also be more efficient in many situations since multiple read access to source code and write access to disk is not needed. The drawback is the need to have all the machines, including the user workstation, sharing the same architecture. We believe this is a very common situation, so this is the default behavior. To increase the loading speed, PAF can be instructed to load the libraries in the slaves using a shared directory. If this is not the case, the compiled code is sent to the PROOF cluster using the native PROOF protocol.

In the event of an infrastructure with computers with different architectures PAF can be configured to compile everything in the worker nodes. All of them will receive all the code, compile and load into their subsystems. These options could be used also in a homogeneous cluster but it is slower and it is, therefore, not recommended in that case.

3.6. Analysis project configuration
Once the analysis code is defined and implemented the user has to instruct PAF about the way it should be executed. To describe that we propose the project paradigm, frequently used in other computing environments.

A PAF project stores the settings of a complete analysis job and can be configured very easily. The most basic project that can be created and launched should only define the selectors to be used and the list of input data files. PAF provides default values for the rest of the possible settings.

Here is a list of configuration handles available in a PAF project:

- Mandatory: The selectors and the order we want to execute them in an analysis.
- Mandatory: The input data files to be analyzed.
• Optional: The execution environment (see next section) where analysis is to be executed.
• Optional: The name of the output file to store the result of the analysis.
• Optional: The input parameters to be passed from the client to the selectors in the worker nodes.
• Optional: The logger to be used
• Optional: The place where the code will be compiled: client or slaves (see subsection 3.5).
• Optional: The possibility to use a shared file system to pass the compiled code.
• Optional: The list of auxiliary dynamic libraries that may be needed by the analysis code.
• Optional: The list of dynamic histograms to be drawn in real time during analysis execution.

4. PAF execution environments
An execution environment describes the type of infrastructure where the analysis will be executed. PAF supports a bunch of different scenarios including the most commonly used like PROOF Lite or PoD. The full list of PAF execution environments implemented includes:

- **PROOF Lite** [5]: This environment uses the client machine taking advantage of modern multi-core desktop computers. By default it creates a process for each core but it can be tuned to use less in order not to saturate the client. It is a good options to speed up a simple analysis if it does not require a lot of CPU and/or data.

- **PROOF on Demand (PoD)** [6]: Developed at GSI, PoD provides several jobs managers through a plug-in system. It is designed to provide an easy configuration of a dynamic cluster over batch systems, grid or ssh accessible clusters.

- **PROOF Cluster** [7]: Also developed at IFCA for CMS data analysis. It uses a fixed master configured and tuned for the specific characteristics of the site where it is installed. The workers however are dynamically configured and started by submitting jobs to a SGE or Torque/PBS batch systems.

- **PROOF Cloud** [7]: Developed at IFCA, PROOF Cloud creates complete PROOF clusters on demand, both master and a set of workers, by starting virtual machines using a IaaS (Infrastructure as a Service) cloud platform.

- **Sequential**: This environment executes the analysis on a single thread. The sequential mode is very useful to debug the analyzers source code without dealing with the problems that may arise in a distributed environment.

As it happens with most of the functionality described in PAF, there is a strategy pattern created that brings users the ability to implement more execution environments. The PAF execution environments are designed to be totally independent from the analysis. Any analysis can seamlessly run in any execution environment with no modification.
5. Performance

We have tested the performance of this new version of PROOF Analysis Framework on HP DL360p Gen8 servers with 2 Xeon E52650 (2.00GHz) connected to a Hitachi HUS 110 storage server. These machines have 32 GB of RAM and 32 execution threads (16 physical cores). We have opted for a typical CMS diboson (WZ) cross section estimation analysis code running over 23 GB of Z+Jets data sample.

In figure 4 we compare the processing time spent in the analysis with and without lazy loading activated in two different execution environments. An improvement of a factor around 3 is observed when the lazy loading of branches is active significantly reducing the total analysis time.

![Figure 4](image)

**Figure 4.** Performance comparison between using two execution environments (PoD and PROOF Lite) with and without lazy branch loading processing a typical CMS diboson (WZ) cross section estimation analysis with 23GB of input data.

In figure 5 we show the speed up with respect to the number of slaves for a PROOF Lite execution environment on the same machines. It can be seen that the choice of using lazy or not-lazy loading does not affect the scalability. Please note that the even though the points are close, the time spent in each case for the two graphs may be different. The observed values scale reasonably for a small number of parallel nodes. The lack of scalability for higher values may be due to the node hitting a limit on the input bandwidth, but a more careful study is needed to draw definite conclusions.
Figure 5. Speed up chart for the cases with and without lazy branch loading for the PROOF Lite execution environment on a HP DL360p Gen8 servers with 2 Xeon E52650 (2.00GHz) and 32 GB RAM connected to a HUS 110 storage server. The scalability with nodes is similar in both cases. Please note that the even though the points are close, the time spent in each case for the two graphs may be different.

Here we have only showed the time taken to execute the analysis in the worker nodes. One of the main time consumers in previous versions of PAF was the time spent preparing PAF, compiling the user code, preparing the execution environment, sending all the code to the workers and loading the libraries. In the new PAF release, many configuration processed have been optimized to reduce that preparation time, lowering it now down to less than 10 seconds for complex tasks. Once the analysis have finished, the time taken to release the resources is negligible.

6. Conclusions and future work
PROOF Analysis Framework is a distributed processing tool designed to cope with the last phases of a typical HEP analysis. It was successfully used to produce the final results for several CMS analysis during the LHC Run-1 data taking. The main gain came from the fact that the researchers had smaller dead-times waiting for results to come leading to a faster analysis development. Some analysis found a factor five or more gain in the overall processing time when adapted to PAF.

In the last year it has gone through a process of re-design and re-implementation to introduce new functionalities, to improve scalability, to add flexibility and to improve its performance in view of the more demanding conditions expected for the LHC Run-2 data taking period.

The introduction of modular capabilities allows for better sharing of analysis algorithms. The lazy branch load mechanism accounts for most of the performance gain with respect to previous versions. Other sources of performance gain come from the new design and the optimization of the compilation tasks. We are now studying how to further reduce the initialization time by adopting even more optimal strategies. PAF flexible structure is able to adapt to complex ROOT tree formats and analysis workflows. We expect to further enhance this aspects in the future.

PAF has been originally developed for ROOT 5.32. The new ROOT 6.x is currently under heavy development. We have already been able to integrate PAF with this new ROOT version and we expect to release it in the near future.
References

[1] Rene Brun and Fons Rademakers 1997 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. 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] See http://www.hep.uniovi.es/PAF

[4] González Caballero I, Rodríguez Marrero A, Fernández del Castillo E and Cuesta Noriega A 2012 A PROOF analysis framework J. Phys.: Conf. Ser. 396 022017

[5] G. Ganis et al. 2008, Proceedings of Science, XII Advanced Computing and Analysis Techniques in Physics Research (ACAT08), Erice, Italy

[6] Malzacher P and Manafov A 2010 PROOF on Demand J. Phys.: Conf. Ser. 219 072009

[7] Rodríguez Marrero A, González Caballero I, Fernández del Castillo E et al. Integrating PROOF Analysis in Cloud and Batch Clusters J. Phys.: Conf. Ser. 396 032091