PROOF - The Parallel ROOT Facility

B. Bellenot, R. Brun, G. Ganis[♯], J. Iwaszkiewicz, G. Kickinger, A.J. Peters, F. Rademakers
CERN, Geneva, Switzerland
M. Ballintijn, C. Loizides, C. Reed, *MIT, Cambridge, MA, USA*
P. Canal, *FNAL, Batavia, IL, USA*
D. Feichtinger, *PSI, Villingen, Switzerland*
[♯]*Contact:* gerardo.ganis@cern.ch

Abstract

The Parallel ROOT Facility, PROOF, enables the interactive analysis of distributed data sets in a transparent way. It exploits the inherent parallelism in data of uncorrelated events via a multi-tier architecture that optimizes I/O and CPU utilization in heterogeneous clusters with distributed storage. Being part of the ROOT framework PROOF inherits the benefits of a performant object-oriented storage system and the wealth of statistical and visualization tools. Dedicated PROOF-enabled testbeds are now being deployed at CERN for testing by the LHC experiments. This paper describes the status of PROOF, focusing mainly on the recent and ongoing developments.

INTRODUCTION

The Parallel ROOT Facility, PROOF, is an extension of the ROOT system [1] aiming to provide users with a tool to run transparently, and interactively, their ROOT analysis jobs on potentially large clusters, hiding the cluster complexity behind the same API and syntax as for local ROOT sessions.

The PROOF principles and benchmark results have been presented before [2]. We just remind here that PROOF achieves its goals by implementing a flexible multi-tier architecture, the main components being the client session, the master server, and the worker servers. The applications running in the master and worker nodes are real ROOT sessions forked via dedicated daemons. Support for running on a network of geographically separated clusters (the *Grid*) is provided by allowing a hierarchy of masters, with a super-master serving as entry point in the system, and the others coordinating the activities of the worker nodes on their respective clusters. The master is responsible for distributing the work using a pull architecture such that worker nodes ask for more work as soon as there are ready. In this way the faster worker nodes are assigned more data to process than the slower ones and load balancing is naturally achieved. The master is also in charge of merging the output it collects from the worker nodes, so that the client receives a single set of output objects, as would have been the case for local processing.

The end-user analysis case addressed by PROOF is the one typically found in HEP experiments, where the data are collection of uncorrelated events. The solution provided by ROOT for this use-case is shown in figure 1. The data are organized in a hierarchical "tree-like" data structure allowing efficient access to only those "leaves" containing information relevant for the analysis. The analysis algorithms are embedded in a framework providing a *selector* template for automatic loop operations on the events; the user needs only to implement functions for the initialization, processing, and termination phases.

With samples containing uncorrelated events, *basic* parallelism is naturally achieved by splitting the whole sample into sub-samples which can be analyzed concurrently.

For the analysis of the forthcoming data, the LHC collaborations foresee [3] the deployment of dedicated facilities consisting of a few *MSpecInts2K* of computing power (currently achievable with a few hundred CPUs) and the data available from a distributed storage system, e.g. CASTOR.

The standard approach to use these facilities is to instrument them with a batch system, e.g. LSF. In this case, the user exploits basic parallelism by splitting a big job addressing the whole data sample, into several sub-jobs, each one addressing a sub-sample of the data, submitted independently and run concurrently, at least to the extent allowed by the policy of the batch system. This *classical* approach provides a somewhat *static* use of resources: any change in the sub-sample definition requires cancellation and re-submission of the corresponding job(s). Moreover, real-time feedback functionality is not automatically provided, unless the job itself is instrumented to do so.

PROOF provides an alternative approach to achieve basic parallelism, resulting in a more dynamic use of the available resources. The job is submitted as a whole and the system takes care of splitting it in *packets* which size depends of the worker performance. To minimize data transfers, workers get assigned as much as possible data replicated on their local disk. This provides a better use of the resources for short and medium term queries, for which the overhead from job preparation, submission, data access and result merging may be substantial for batch systems. In addition PROOF provides continuous real-time monitoring on the running jobs and better control on the tails in the CPU-time distribution by allowing faster workers, once idle, to take over the packets assigned to slower workers. For long term queries, we expect PROOF to be sligthly less efficient than the batch approach due to the overhead of distributing packets (the corresponding inefficiency is estimated to be of the order of 5-10%); the advantages of real-time feedback and CPU-time "tail" control will, however,

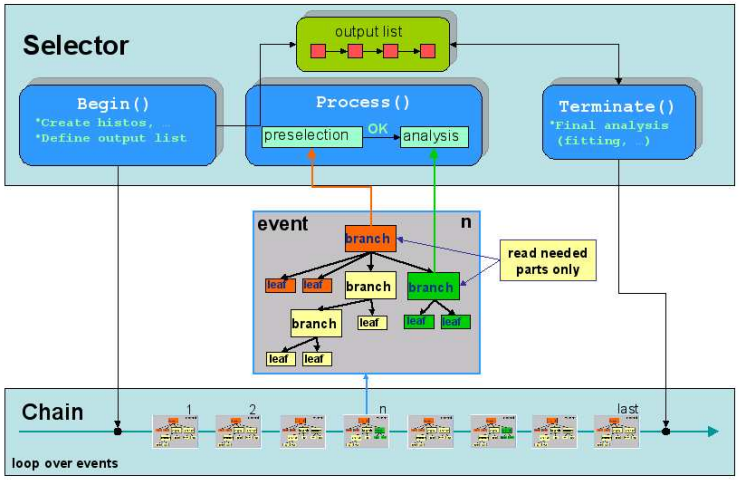

Figure 1: The ROOT model for data analysis.

remain.

During the last few years, PROOF has emerged from the status of proof-of-concept to a status where experiments can start using it for real work. In view of the approaching of LHC real data, in Spring 2005 the ROOT team has decided to start a *new* PROOF *development cycle*. The initial goal of this effort was to identify the main issues to be addressed to bring PROOF to production level.

The remaining of this paper is devoted to the discussion of these issues, starting with those for which a solution has been already implemented.

RECENT ADDITIONS

Background mode and Stateless connection. In the original design, PROOF was intended for strict interactive usage: the idea was to bring medium term jobs (around an hour) to a perceived length of a few minutes, typical of the short queries usually accepted for interactive work. For this reason, submitting a query blocked the ROOT session in the same way as, for example, the processing of a tree resident on the local hard disk did. However, the exact definition of *few minutes* is subjective, and the user, who is aware that the local machine is basically idle, may find more convenient to continue processing the query in the background, or even to detach from the session and come back later to retrieve the results. These two options are now available.

Support for the *background* mode has required a modification of communication protocol between the client and the master to allow for asynchronous collection of the messages coming from the master in reply to client requests. These modifications have also allowed to remove the limitation to one PROOF session per ROOT shell, originally present: it is now possible to operate concurrently PROOF sessions on different clusters.

New communication layer based on XRD. Support for disconnect/reconnect functionality has required a deep change in the underlying connection layer on which the cluster relies. The main reason is that in the original PROOF the connection Client-to-Master was part of the session state, strictly interconnecting the session and the connection lifetimes. The solution that we implemented required the introduction of a new logical component, the *PROOF server coordinator*, keeping track of alive sessions and representing the entry point to the cluster. After some investigation we realized that *xrd*, the main component of the XROOTD daemon [4], met most of the requirements of the coordinator rôle in terms of connection handling and message dispatching. As XROOTD is already distributed with ROOT, we have designed and implemented a new *xrd* protocol plug-in (*XrdProofdProtocol*) allowing a user to start or reconnect to a PROOF session via *xrd*.

The new connection layer is shown in figure 2. Each node runs an instance of the server coordinator; the coordinator is in charge of forking a new PROOF server session (*proofserv* or *proofslave*) and to control the message flow between Client/Master or Master/Workers, dispatching the messages to the appropriate recipient. For each new user, the coordinator creates internally an instance of *XrdProofdProtocol*; this instance owns the connections to the client and to the PROOF servers associated to the user. *XrdProofdProtocol* allows for many client-like connections, so that a client can connect to the same session from different places, e.g. from the pit to control the ongoing processing. Client connection can be closed and open at any moment, even during processing. The overhead introduced by the additional components has proven to be negligible.

The *XrdProofdProtocol* can be run concurrently with the standard *xrd* protocol used for data serving. This makes XROOTD the ideal candidate to manage the pool disks local to the PROOF resources. Dedicated XROOTD configurations are being designed to optimally exploit this possibility.

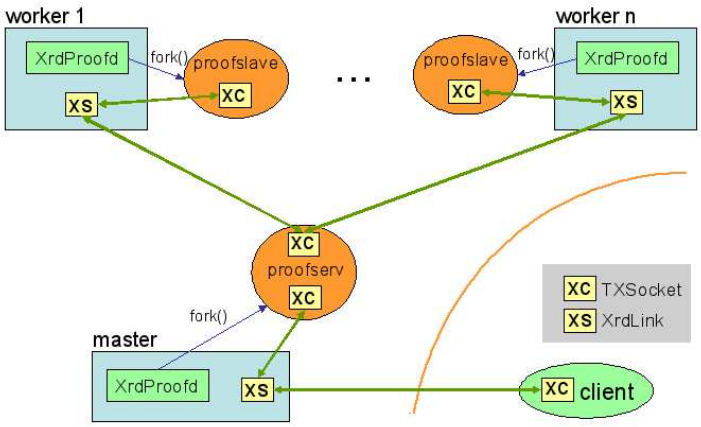

Figure 2: The new connection layer based on XROOTD [4]

Query handling. Concurrent query processing requires a proper classification of the results of queries. For this purpose we introduced a new class containing all the information about the query, including the selector macro in compressed form. Instances of this class for each query run in the current or previous session are kept on the master and can be retrieved - or archived to a ROOT file - at any moment. A complete set of methods to handle the results of queries has been added to PROOF client API.

Data-set handling. A *data-set manager* to handle the definition, preparation and classification of the set of data files to be analayzed, was another required addition. We have recently implemented tools to facilitate the upload of a set of files on the disk pool of the PROOF cluster. These tools use the logical concept of *data set* to identify a set of files; the meta-information about a data set is stored on the master and can be retrieved at any time. Scripts to automatically stage out data files via XROOTD are also available.

Session viewer. To deal with the increased functionality of the PROOF API we developed a powerful graphical user interface. Figure 3 shows a snapshot of the current status of the GUI. The full PROOF client API can be controlled via the session viewer. It is possible to select and run scripts defining the data sets and prepare query submission navigating in the appropriate directories. Feedback from running queries can be monitored in real-time, by switching between query contexts with a simple click.

ONGOING DEVELOPMENTS

The main issues on which we are currently focusing are the following: efficient scheduling in multi-user environment, access optimization to remote data and robustness increase.

Multi-user issues. PROOF has been shown to perform pretty well in scenarios where the number of users was limited to a few. However, when the number of users on the assigned workers is such that the PROOF servers use a significant fraction of the available resources, the many sessions interfere with each other triggering continuous page swaps considerably deteriorating the performances. It is therefore necessary to introduce a way to control the use of the available resources to guarantee the best response to all users according to their priorities.

For this purpose we are developing a new component - the *scheduler* - in charge of assigning resources to processes representing different users basing the decisions on the information about system load and in accordance with the scheduling policy. The metric used to evaluate the priority of a certain query will be based on information on the resources still needed by the query to be terminated, the performance of the assigned workers, the amount of I/O, the effective bandwidth per data server and the user profile (history of previous submissions).

The new infrastructure based on *xrd* allows also more flexibility in accessing and dispatching the required information and in modifying on the fly the set of workers assigned on the query. The idea is that the scheduler will regularly inquire the masters for the status of their queries and dispatch instructions on how to proceed during the next *quantum* of time.

Data Access issues. Although not specific to PROOF, another problem already mentioned is the one of efficient data access. In general, HEP analysis tend to be I/O bound. However, at LHC, due to the large average number of tracks in the events, the fraction of time spend in data elaboration may, in general, be important.

As discussed in detail in [5], in such a case, using caching techniques in conjunction with asynchronous prefetching techniques, it is possible to reduce significantly the latency in accessing a data segment. Under certain circumstances, depending on the fraction computing time and the effective network bandwidth, it is possible to reach latency values similar to those observed for local access.

Additional improvements could come from the detailed knowledge of the segments needed by the analysis job in

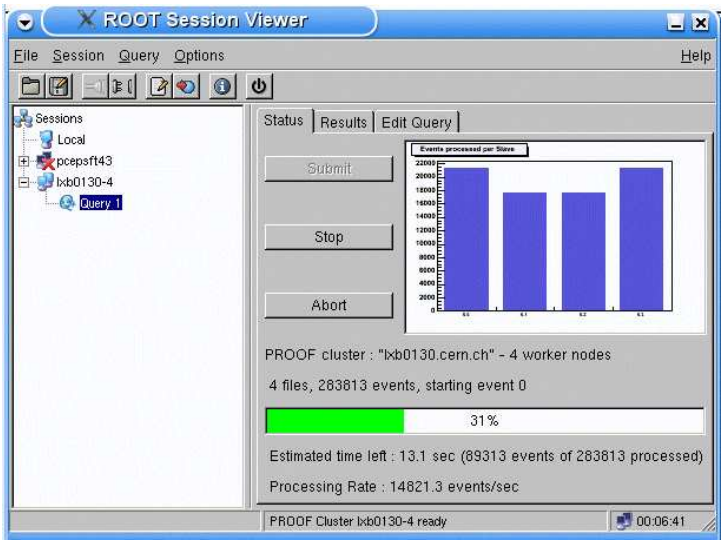

Figure 3: The session viewer can be used to control multiple PROOF sessions.

the next steps. This information is available from the tree headers. The integration of the techniques discussed in [5] with the ROOT tree information is in progress.

Robustness. To be reliable, a complex distributed system needs to react to fault situations in a rock-solid way, allowing the client to always have the situation under control. The protocol of the XROOTD client-server connection allows already more flexibility in treating error conditions. In addition we plan to fully exploit the parallel control network provided by XROOTD, to be able to monitor continuously and independently the status of the PROOF servers and also of the coordinators, and to take the appropriate actions.

AoB. Current activities also include the development of an operations monitoring system using MonALISA [6], for which an advanced prototype is being tested, and the addition of worker self-announcing functionality, which could also be used to implement a certain degree of self-healing.

Test-bed facilities. These developments are tested on dedicated testbeds provided by CERN/IT. Larger test analysis facilities are being setup at CERN and IN2P3-Lyon which will be instrumented with PROOF for testing and validation mostly for the ALICE and CMS experiments.

PROOF is being used since quite sometime at MIT for the analysis of PHOBOS data [7]. Recently interest has also raised within the CMS collaboration [8].

CONCLUSIONS

The PROOF system provides an alternative approach for end-user analysis on the foreseen Tier 2 /Tier 3 facilities with the aim is to improve the response and transparency for short and medium term queries. There is also a plan to use PROOF for prompt analysis of hot channels on the large Central Analysis Facility foreseen at CERN [3].

In the last year a large effort has been put in the project. A new connection layer has been introduced base on XROOTD and many new functionalities added to the API. While several important development are still in progress, test facilities have been setup at CERN and start to be used by the experiments. This is expected to provide the necessary feedback to bring the system to the level of robustness, flexibility and user-friendness needed by a real analysis facility.

REFERENCES

[1] René 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. See also http://root.cern.ch/.

[2] M. Ballintijn *et al.*, *The PROOF Distributed Parallel Analysis Framework based on ROOT*, Proc. CHEP03 Intl. Conf., La Jolla, California, Mar. 2003, http://arxiv.org/abs/physics/0306110; M. Ballintijn *et al.*, *Super Scaling PROOF To Very Large Clusters*, Proc. CHEP04 Intl. Conf., Interlaken, Switzerland, Sep. 2004.

[3] For the computing models of the LHC experiments see: ALICE Coll., CERN-LHCC-2005-018; ATLAS Coll., CERN-LHCC-2005-022; CMS Coll., CERN-LHCC-2005-023; LHCb Coll., CERN-LHCC-2005-019; LCG Project, CERN-LHCC-2005-024.

[4] http://xrootd.slac.stanford.edu

[5] See presentations at this conference by F. Furano, abstract #368, and A. Hanushevsky, abstract #407 .

[6] http://monalisa.cacr.caltech.edu/monalisa.htm

[7] M. Ballantijn, presentations at this conference, abstract #374

[8] I. Gonzalez, presentations at this conference, abstract #267