Parallel computing of ATLAS data with PROOF at the Leibniz-Rechenzentrum Munich

Philippe Calfayan\textsuperscript{1}, Matthias Schott\textsuperscript{2}
\textsuperscript{1}Ludwig-Maximilians-Universität Munich, Germany
\textsuperscript{2}CERN, Switzerland
E-mail: philippe.calfayan@physik.uni-muenchen.de

Abstract. The Parallel ROOT Facility (PROOF) has been exercised at the Leibniz-Rechenzentrum Munich (LRZ, Tier-2) with ATLAS data. Tests of scalability have been carried out, and different storage strategies for the input files have been confronted. The performance of distributed analyses coded in Python and compiled C++ has been compared when running on input files in ATLAS pool format.

1. Introduction
The PROOF [1] (Parallel ROOT [2] Facility) libraries are ROOT extensions for parallel data processing at event level. It is integrated in the ROOT analysis framework and can run interactively with the possibility of having a heterogeneous cluster of computers. With the forthcoming start-up of the Large Hadron Collider, the ATLAS experiment will have to take up the challenge of processing huge amount of data. Since the collisions recorded are independent, ATLAS analyses could highly benefit from the parallelization provided by PROOF. Figure 1 represents the architecture of a PROOF cluster in the case where a single master node orchestrates several worker units. Configurations including multiple masters have not been considered in the present study.

2. The PROOF installation at the Leibniz-Rechenzentrum
A PROOF cluster hosted at the Leibniz-Rechenzentrum consisting of a scalable amount of up to ten worker nodes (Opteron) has been exploited in order to conduct performance tests in the case of interactive ATLAS analyses. Each worker node provides four processing cores running at 2.6 GHz and is associated with 8 GB of RAM.

Scenarios of various complexities have been considered to exercise PROOF with ATLAS data and evaluate its utilization in real life conditions. The investigation of the PROOF performance at the LRZ focused on: varying the number of parallelized processing units (CPU cores), the amount of simultaneous users, and the type of file storage.

Figure 2 illustrates the two approaches that have been examined in order to exercise the PROOF cluster. Two input file formats are distinguished, namely the ATLAS pool format, referred to as \textit{AOD}, and the native ROOT format, referred to as \textit{D\textsuperscript{3}PD}, which is commonly adopted to store the output resulting from skimming of the \textit{AOD} files. \textit{D\textsuperscript{3}PD} and \textit{AOD} files gather events with sizes of nearly 1 to 10 kB and 100 kB, respectively.
The test analysis is based on a ROOT event loop which was compiled in C++. To access the content of AOD files from ROOT, the use of a specific wrapper from the ATLAS Athena framework is required, namely the package AthenaROOTAccess (ARA). This tool, which utilizes Python code, converts the persistent tree that is included in AOD files into a ROOT transient tree, stored in memory. When running on AOD input files, the PROOF analysis loop, designed for C++, was either coded in Python, using the TPython wrapper provided by ROOT, or in C++.

**Figure 1.** Illustration of the architecture deployed for a PROOF cluster, in the case of a single master configuration.

**Figure 2.** Representation of the two methods that have been considered to test PROOF at the LRZ. Input files in either native ROOT or ATLAS pool formats. Analyses based on compiled C++ and Python have been considered.
3. Comparison of storage strategies

Different strategies have been evaluated for the storage of the input data files. We distinguished between the three following cases:

- The data are stored on each local node, using their local disks.
- The data access is operated via client/server connections to a dCache [3] storage element, which is composed of disks combined with a RAID 6 array and served through a 10 GB switch.
- The data are located on disks formatted with the Lustre network clustering filesystem [4], which is optimized for parallel computing. As in the case where the local disks of the nodes are exploited, all workers can access the data without a dedicated server.

A simple test analysis, based on the Z boson reconstruction, and on the generation of control histograms, is processed via a PROOF event loop, using ROOT version 5.20. A complex variant includes 200000 hyperbolic tangent operations per event. The simple and complex analyses are intended to be dominated by the data transfer (I/O) rate and the CPU resources, respectively. Input data files are in the $D^3PD$ format, and contain 1.6 million of events with a size of nearly 4 kB.

The total execution time includes the initialization of the PROOF cluster, the delay due to the data transfer rate, the actual event processing, and the merging of the analysis output. The following results do not take the initialization time into account. The speedup factor $S$ describes the gain of processing time $T_n$ when requesting $n$ parallel cores compared to the utilization of one single core. It is defined by: $S = \frac{T_1}{T_n}$.

![Figure 3. Speedup factor of a CPU-dominated (complex) analysis (left) and I/O-dominated (simple) analysis (right) running on input files in native ROOT format, as function of the number of cores available in the PROOF cluster. Storage strategies based on local files, Lustre and dCache are compared.](image)

In the right plot of Figure 3, the storage strategies are confronted with running the simple variant of the test analysis. Events are processed by PROOF using a CINT dictionary [2]. The discrimination between the storage alternatives is represented in terms of speedup factors. The
scalability of the PROOF cluster turns out to be limited by the data transfer rate of the storage systems, and comparable performances are found for all the storage strategies tested, with a slight disadvantage for dCache.

When carrying out the tests with the complex analysis, the I/O related limitations of the storage elements do not prevail, and a better scalability is observed, as highlighted by the left plot of Figure 3.

4. Multi-user application
A realistic use of PROOF would imply the management of multiple users simultaneously. Tests are conducted with the same installation as in Section 3. Only one PROOF cluster has been set up. Each user opens a new session on the same cluster.

The analysis utilized for these tests is the complex variant of the one in Section 3, so that the influence of the data transfer rate can be neglected. The Lustre filesystem has been chosen, and it is assumed that all users perform their analyses by taking advantage of all available cores \((n = 40)\). Effects of potential file caching have not been avoided. Having \(U\) users, the speedup \(S\) is expected to be divided by \(U\) with the time \(T\) being longer by a factor \(U\). Figure 4 confirms the predicted scalability. When \(U > 1\), the time \(T\) and the factor \(S\) that are shown are averaged over \(U\).

![Figure 4](image)

**Figure 4.** Processing time and speedup factor of a CPU-dominated analysis running on input files in native ROOT format, as function of the number of concurrent users (independent PROOF sessions). A total of 40 cores has been used for this test. Average processing times and speedup factors are shown.

5. Performance when running on ATLAS pool files
The performance of PROOF has been tested with ATLAS pool files as input, using a total of nearly 12500 \(W \rightarrow \mu \nu\) simulated events generated with Athena v14.2.20 \(\sqrt{s} = 10\,\text{TeV}\). These input files were stored with the Lustre filesystem.

The processing of AOD input files with PROOF via an analysis loop compiled in C++ cannot be achieved in association with a CINT dictionary, since the latter is unable to handle
the complex C++ code used in the ATLAS pool libraries. The access to the ATLAS pool classes is nevertheless made possible by the utilization of a REFLEX dictionary [2], which is able to manage advanced C++ structures.

We compiled the test analysis with the dedicated ATLAS environment and generated the corresponding REFLEX dictionary, using the ATLAS Athena release v14.2.23. The analysis loop, which calculates the $W$ transverse mass for 10000 times and plots control histograms, is designed to be dominated by CPU resources. Two versions are considered: one is based on a compiled C++ event loop, while the other one reads the ARA transient tree with Python (via TPython).

Comparable performances are obtained for both versions of the analysis in the case where the calculation of the transverse mass is not repeated. Figure 5 illustrates the contrast between both programming languages, in terms of speedup factors. The compiled C++ version reaches the I/O limit earlier than its Python counterpart since it has better CPU performance.

![Figure 5](image.png)

**Figure 5.** Processing time (left) and speedup factor (right) of a CPU-dominated (complex) analysis running on ATLAS pool input files, as function of the number of cores available in the PROOF cluster. The use of either compiled C++ or Python code is compared.

6. Conclusion
The PROOF libraries turn out to significantly improve the performance of the interactive ROOT based analyses in a scalable way. The limitations due to the data transfer rate of the storage systems tested, however, point out that large clusters provide scalable performance especially when considering complex calculations, which fits the requirements of ATLAS analyses using advanced algorithms. Nonetheless, in our tests, the performance of the storage system could be increased, for instance, by using advanced hardware such as solid state disks, which allow ultrafast data access.

In addition, the flexibility offered by REFLEX dictionaries for the use of compiled C++ analyses shows that PROOF in association with AthenaROOTAccess is well suited for an interactive use with typical ATLAS data formats.
References
[1] M. Ballintijn, et al, Nucl.Instrum.Meth. A559, 13 (2006)
[2] R. Brun and F. Rademakers, An object oriented analysis framework (http://root.cern.ch)
[3] M. de Riese, et al, The dCache Book (http://www.dcache.org/manuals/Book/)
[4] Lustre collaboration, Lustre state and production installations (http://wiki.lustre.org)