CERN Summer Student Programme 2014

Work Report

Parallelizing TTree::Draw functionality with PROOF

Author:
Stefano Marinaci

Supervisor:
Gerardo Ganis

September 19, 2014

Introduction

Multicore processors have been available in commodity computers since quite a while. However, it was generally not possible to take real advantage of all the available CPU for I/O intensive tasks, due to the bandwidth limitations of standard hard drives. Today's laptops and desktops, in addition of having processors with 4, 8 or more cores, are also equipped with large amounts of RAM and, more and more often, with SSD drives. With these devices it becomes possible to exploit the full potential of all cores also for I/O bound tasks. To exploit multicores there are basically two possibilities:

- **Threads of execution**: small sequences of programmed instructions that work and can be managed independently.
  Typically are used for parallelization of a program to take advantage of modern multi-core processors: in fact, a single thread can run on only one core at a time. Threads are the optimal way of exploiting multi-cores but require thread-safeness and thread-awareness of the software.

- **Multi-process programming** exploits the outcome of multiple concurrent processes to improve performances.

In ROOT[1], the software context of this project, multi-threading is not currently an easy option, because ROOT is not by construction thread-aware and thread-safeness can only be achieved with heavy locking. Therefore, for a ROOT task, multi-processing is currently the most effective way to achieve cuncurrency. Multi-processing in ROOT is done via PROOF[2]. PROOF is used to enable interactive analysis of large sets of ROOT files in parallel on clusters of computers or many-core machines. More generally PROOF can parallelize tasks that can be formulated as a set of independent sub-tasks. The PROOF technology is rather efficient to exploit all the CPU's provided by many-core processors. A dedicated version of PROOF, PROOF-Lite, provides an out-of-the-box solution to take full advantage of the additional cores available in today desktops or laptops.

One of the items on the PROOF plan of work is to improve the integration of PROOF-Lite for local processing of ROOT trees. In this project we investigate the case of the Draw functionality and its implementation via PROOF.

1. The Draw functionality

One of the most used functionalities in processing ROOT Trees is "Draw". The "TTree::Draw"[3] method is used for example, for:

- Variable plotting;
- Histogram or ad-hoc output object creation;
- Selection criteria design and optimization;
- ...

The draw functionality is implemented using TSelectorDraw[4] which is a realization of the generic TSelector "abstract" class.
The process flow is shown in Figure 1.
When the user invokes the Draw method, the tree initialize its internal engine TTreePlayer[5] and invokes the DrawSelect method of the player which in turn does the processing using TSelectorDraw. The output of process is an histogram which then is displayed in the canvas seeing by the user.

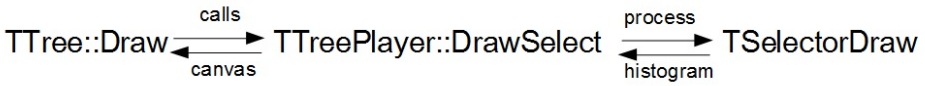

Figure 1: Show how the draw function works

1.1 TSelector framework

TSelector is the framework proposed by ROOT for analyzing event like data organized in trees. The user derives from the TSelector class and implements the member functions with specific analysis algorithms. When running the analysis, ROOT calls the member functions in a well defined sequence and with well defined arguments. By following the model this analysis class can be used to process data sequentially on a local workstation and in batch or in parallel using PROOF.
TSelector is provided by ROOT to process trees and is basically structured in three main parts:

- **Begin()**: is called once before the loop on the tree starts.
  A convenient place to create histograms;
- **Process()**: is called for every entry in the tree.
  Used to read the entries and fill histograms and/or other output objects;

- **Terminate()**: is called at the end of the loop on the tree.
  A convenient place to draw histograms.

PROOF requires the tasks to be formulated in the TSelector framework, parallelizing the Process step, assuming the entries be uncorrelated.

1.2 Draw in PROOF

PROOF already provides a sub-sample of the "Draw" functionality via the TProof::DrawSelect method and a set of specialized selectors[6] shown in Table 1.

TSelector
TProofDrawEntryList
TProofDrawEventList
TProofDrawGraph
TProofDrawHist
TProofDrawListOfGraph
TProofDrawListOfPolyMarker3D
TProofDrawPolyMarker3D
TProofDrawProfile
TProofDrawProfile2D

Table 1: complete list of specialized TSelector used currently by PROOF

TProofDraw has some of the functionality implemented in TSelectorDraw but is not complete, for example it misses macro processing or special keyword $ interpretation[1]. In the meantime TSelectorDraw continued to evolve. For functionality completeness and maintenance reasons it is better to keep only one selector for clarity. Since TSelector acts as reference it makes sense to use it also in PROOF. However, TSelectorDraw is not usable in PROOF as it is because it was developed to work in ROOT with only one worker and makes assumptions valid only locally. In particular:

- The availability of the TTree at constructor level (before start processing);
- The possibility to avoid using the output list for the output transfer.

The way PROOF works cannot be changed; what can be tried is to make the local flow similar to the PROOF one. This was the main objective of this project.
In Figure 2. we compare the local flow with the PROOF one. The main

[1] TProofDraw was developed as part of a technical student project; it was intended to replace TSelectorDraw but the works was never finished

differences are in the moment in which the formula is compiled and in the way the output is passed back to the user.

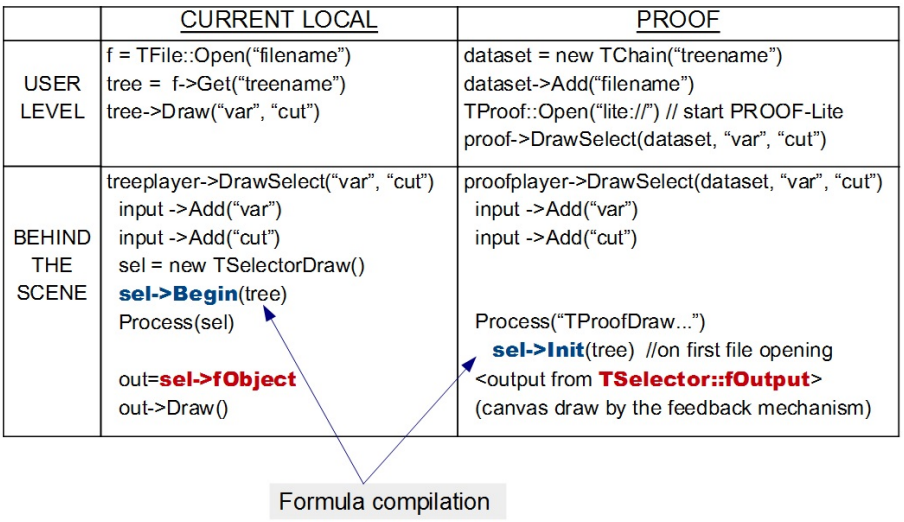

Figure 2: Comparison of the ways the draw functionality is implemented: on the left the local version, on the right the PROOF version

1.3 Changes to TSelectorDraw

To improve the capability of TSelectorDraw the variable formula initialization was moved from Begin to Init and also the output object was made available in the output list.
The changes have been made paying attention to object ownership in list in order to prevent conflicts between current directory (gDirectory) and the selector output list; in fact, to avoid double deletions, only one list should own the object.
Also it was guaranteed that a proper initialization is always run. In particular in the case of multiple draw calls the constructor was not re-run by the TTree::Draw call.

1.4 Current status and future work

- A branch in github repository with all the changes in TSelectorDraw has been created:

`https://github.com/smarinac/root/tree/v5-34`

All changes tested in "roottest" have been passed successfully.

- A PROOF interface implementing Draw in PROOF using TSelectorDraw has been implemented. It works for basic functionality, for example plotting simple variables or using standard math functions. Currently it is in the form of a macro, in the future it will be integrated in the code. The change should be done in TTreePlayer:Drawselect for transparent availability in TTree::Draw but this modification will require some brainstorming and architectural work.
  In addition, missing functionality, such as support for user defined functions, needs also to be activated.

2. Remarks about the original proposal

The original proposal was entitled "Analysing trees in PROOF with TTreeReader". After the submission, some changes occurred in the interface of TTreeReader which made its integration in PROOF automatic. Therefore it was decided to change the assignment by choosing the next item on the PROOF development list, the improvement of the Draw functionality integration.

References

[1] `http://root.cern.ch/drupal/`.

[2] `http://root.cern.ch/drupal/content/proof`.

[3] `http://root.cern.ch/root/html/TTree.html#TTree:Draw`.

[4] `http://root.cern.ch/root/html/TSelectorDraw.html`.

[5] `http://root.cern.ch/root/html/TTreePlayer.html`.

[6] `http://root.cern.ch/root/html/TProofDraw.html`.