Contents

Chapter 1

A PROOF Guide For Beginners

Abstract:

PROOF is a framework to process ROOT trees in parallel using a pull architecture for work distribution. The main underlying assumption being the embarrassing parallel nature of the task at hand, PROOF can also be used for any other task of such a nature, including full or toy simulations or mathematical evaluations. This introductory guide illustrates the main features and usage of PROOF, introduces the relevant terminology and dissects a number of meaningful examples.

Chapter 2

Motivation and Introduction

The primary goal of PROOF is to speed-up tree processing by doing it in parallel. The data of many HEP experiemnts are stored as entries in a TTree structure. These entries are independent and the order of processing is irrelevant for the final result, so the paradigm of embarrassing parallelism applies and we can just split the input sample in many parts, have them analysized by parallel processes and merge the results of each single process.

There is nothing much new so far: splitting and merge has been the way to speed-up analysis, for example on LSF, already at LEP times. What is different in the PROOF approach is the splitting technique, not based anymore on a push architecture (split a priori in usually equal parts), but on a pull architecture, where the splitting is based on the effective performance of the actors. This requires the introduction of a new component, the master, in interactive control of the workers.

PROOF realizes the splitting and merge paradigm using a 3-tier architecture: client, master and workers (see Figure 1.1).

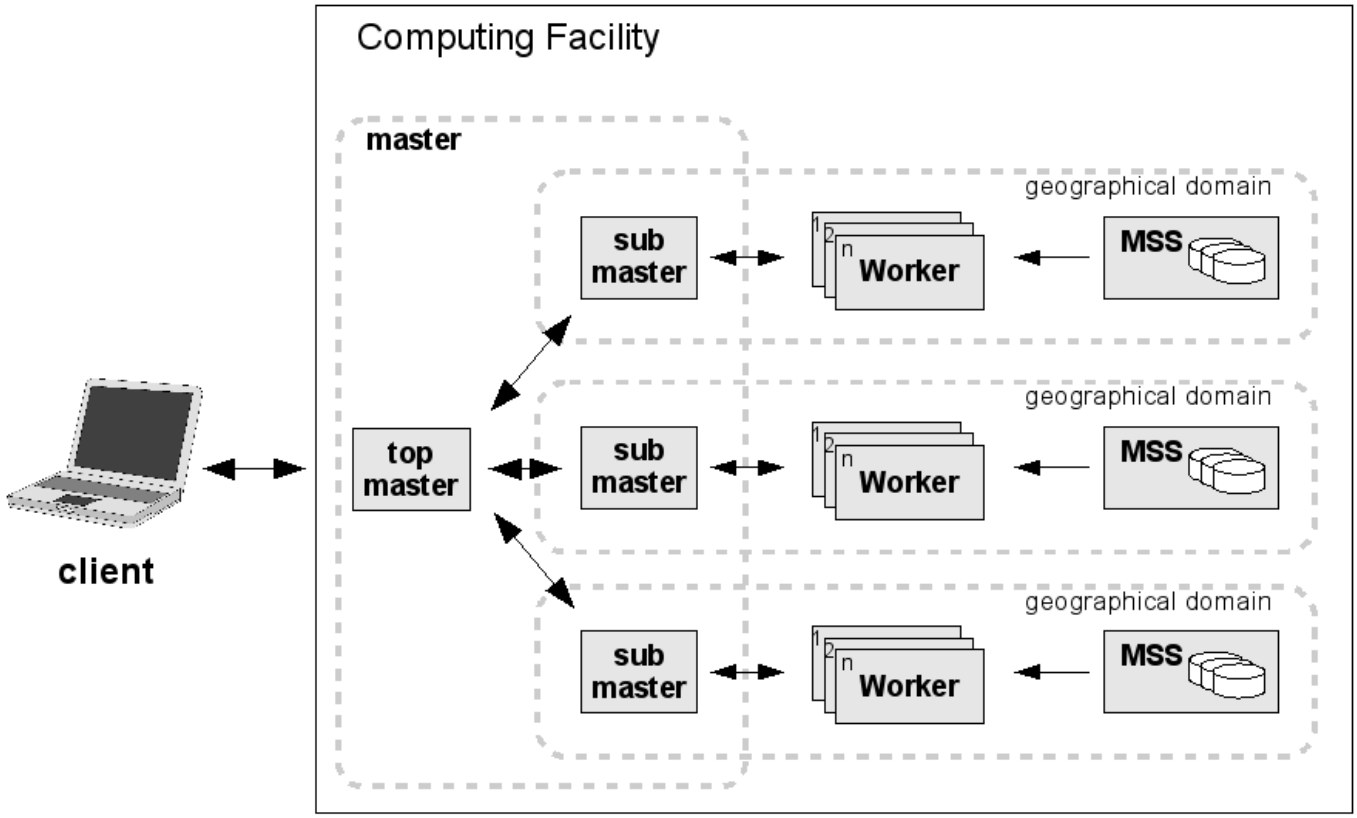

Figure 2.1: PROOF architecture.

The second tier, the master, can be split in additional tiers to connect geographically separated PROOF clusters or to reduce the control work on a single master for large number of workers, improving overall scalability.

The architecture shown in Figure 1.1 reflects the fact that a PROOF-enabled facility is usually distributed over many physically separated nodes, which can be physically distant from the client or user machine. The PROOF processes are

then network connected and a network service provided a dedicated daemon - called *xproofd* - insures that the relevant connections are established.

While this architecture can be adapted to all cases, for single multi-core machines, e.g. desktops or laptops, network-connecting the processes is not strictly necessary. Processes can communicate efficiently using, for example, *unix* sockets. Furthermore, the client tasks can be done directly by the client, removing some a layer of data transfers. The resulting 2-tier architecture is shown in Figure 1.2

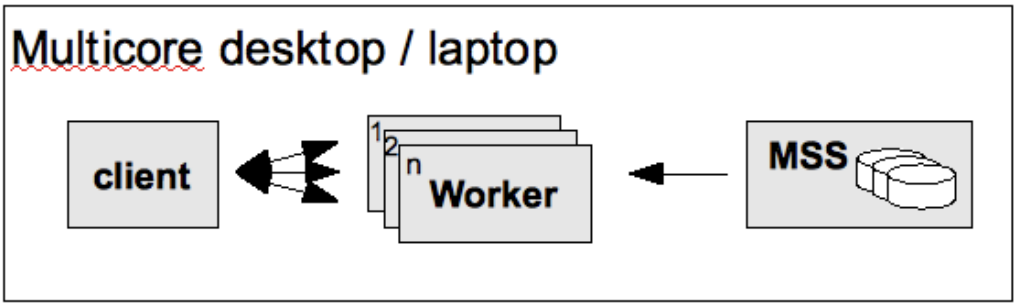

Figure 2.2: PROOF-Lite architecture.

and is implemented in the version of PROOF optimized for multi-core desktops or laptops referred to as PROOF-Lite.

2.1 Welcome to PROOF

The purpose of this document is to serve as a beginners guide and provides extendable examples for your own use cases. This guide will hopefully lay the ground for more complex applications in your future scientific work building on a modern, state-of the art tool for data analysis.

This guide in form of a tutorial is intended to introduce you to the ROOT package in a few handfuls of pages. This goal will be accomplished using concrete examples, according to the "learning by doing" principle. Being PROOF a ROOT application, many basic concepts are the same as in ROOT. The ROOT Users Guide (Team 2013a) and navigate through the Class Reference (Team 2013b) are certainly good refernces to have at hand when doing this tutorial.

PROOF, like ROOT, is written in and heavily relys on the programming language `C++`, and therefore some knowledge about `C` and `C++` is required. Eventually, just profit from the immense available literature about `C++` if you do not have any idea of what object oriented programming could be.

PROOF is part of ROOT and is available for Unix platforms (Linux, Mac OS X, ...), but in this guide we will implicitly assume that you are using Linux. The first thing you need to do with ROOT is install it, don't you? Obtaining the latest ROOT version is straightforward. Just seek the "Pro" version on this webpage http://root.cern.ch/drupal/content/downloading-root. You will find precompiled versions for the different architectures, or the ROOT source code to compile yourself. Just pick up the flavour you need and follow the installation instructions.

Before start divign into the details we introduce some terminology which you will encounter often when working with PROOF.

Chapter 3

PROOF basics

3.1 PROOF-Lite session

3.1.1 TProof: the PROOF shell

3.1.2 The PROOF sandbox

3.2 First processing

3.3 The dialog max

3.4 The output list

3.5 Feedback

3.6 Example of TTree processing: the H1 example

3.7 Datasets

3.7.1 Datasets in PROOF

3.7.2 A dataset for the H1 sample

3.8 Loading additional code

3.8.1 PAR files

3.8.2 PAR example: event.par

3.9 Terminology

Client
The *client* is the machine running the ROOT session opening the connection to the master or to the workers (in case of PROOF-Lite). We call *client* also the client ROOT sessions itself.

Master
The *master* is the machine running a ROOT application coordinating the work between the workers and merging the results. We call *master* also the master ROOT application itself.

Worker or Slave
The *worker* (or *slave*) is any machine running a ROOT application doing the actual work. We call *worker* (or *slave*) also the worker ROOT application itself.

PROOF session
A *PROOF session* is a set of {client, master, workers} started successfully by a call to `TProof::Open()`.

Query
A *query* is a process request submitted by the client to the master. It consists of a selector and possibly by a dataset (chain).

Package, PAR file
A *package* or *PAR file* is additional code needed by the selector, not available on the PROOF cluster nodes, loaded as a separate library. It is given in the form of a gzipped tarball containing all what needed to build and enable the package.

Selector
A *selector* is a class derivign from `TSelector` and providing the code to be executed.

Dataset
A *dataset* is the meta-information for the files containing the `TTree` to be processed. It can be a `TChain`, a `TFileCollection`, a `TDset`. It can also be the name of a `TFileCollection` stored on the master.

Now *let's dive into PROOF!*

Chapter 4

Tutorials

In this section we dissect the tutorials available under $ROOTSYS/tutorials/proof .

4.1 Simple

`Machine that`

4.2 Event

`MCHINE THAT`

4.3 Pythia8

This tutorial shows how we can use PROOF to simulate physics events with the PYTHIA8 generator based on main03.cc example in Pythia 8.1. To run this analysis, Pythia8 must be installed on the client and workers machine and ROOT must be configured with pythia8 enabled.

4.3.1 Installation of Pythia8

The first thing we need is to install Pythia8; this step can be skipped if Pythia8 is already available on the machine(s). Keep anyhow track of the installation paths. For the installation, make sure that you have HepMC installed (available for installation in every major linux distribution) and then download the gzipped tarball from the Pythia web site:

```
$ mkdir -p local/pythia8/par
$ cd local/pythia8/par
$ wget http://home.thep.lu.se/~torbjorn/pythia8/pythia8180.tgz
$ cd ..
$ tar xzf tar/pythia8180.tgz
$ cd pythia8180
$ ./configure --installdir=/opt/pythia8 --enable-shared --enable-64bits
$ make
$ sudo make install
```

This installes under `/opt/pythia8`.

4.3.2 Enabling Pythia8 in ROOT

For the tutorial we need the ROOT Pythia8 plug-in, `libEGPythia8.so`. To build this plug-in we need to enable it at configuration

```
$ cd $ROOTSYS
$ ./configure *other options* --enable-pythia8
$ make
```

For non *standard* include and libraries paths for the Pythia8 installation, we need to specify them with the `--with-pythia8-incdir` and `--with-pythia8-libdir` switches.

4.3.3 Setting the right environment

To run the tutorial we need to set the environment variable `PYTHIA8` to point to the Pythia8 installation directory, i.e.

```
$ export PYTHIA8=/opt/pythia8
```

for the example above. If the file `Index.xml` is not under the default directory `$PYTHIA8/xmldoc` then the variable PYTHIA8DATA must point to directory containing `Index.xml`.

The tutorial assumes that the location of Pythia8 and of Index.xml on the worker machines is exactly the same.

4.3.4 Running the tutorial

To run the tutorial in, for example, PROOF-Lite, just type

```
root[] .L tutorials/proof/runProof.C+
root[] runProof("pythia8", "lite://")
```

You should get the output shown in Figure 4.1

4.4 Concluding Remarks

This is the end of our guided tour for beginners through PROOF. There is still a lot coming to mind to be said, but by now you are experienced enough to use the ROOT documentation, most importantly the **PROOF pages**, but also the **ROOT home page** and the **ROOT reference guide** with the documentation of all ROOT classes.

A very useful way for you to continue exploring PROOF is to study the examples in the sub-directory `tutorials/proof` of any ROOT installation.

End of this guide ... but hopefully not of your interaction with PROOF !

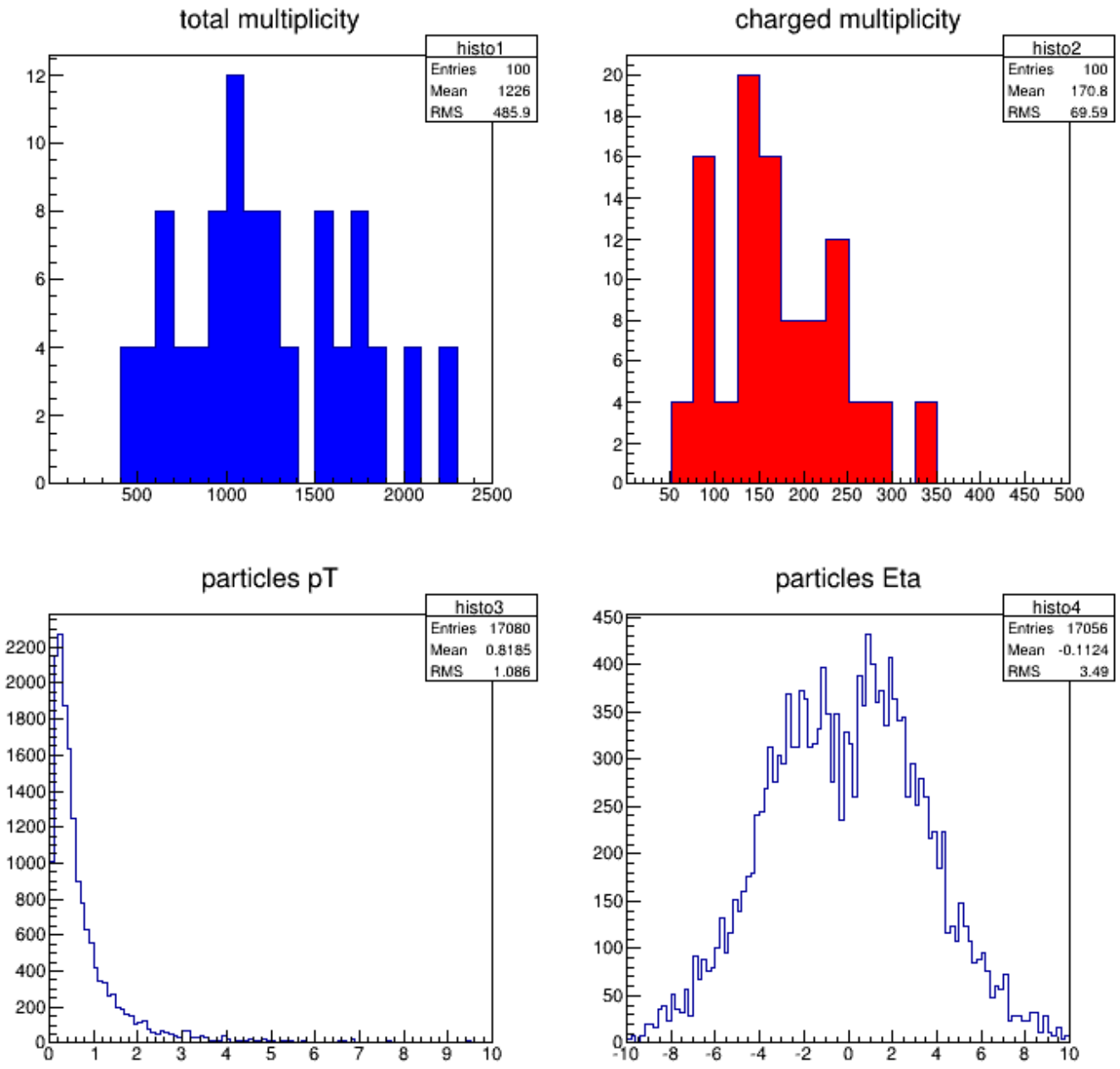

Figure 4.1: Graphical output of the pythia8 tutorial.

Chapter 5

References

Team, The ROOT. 2013a. *The ROOT Users Guide*. http://root.cern.ch/drupal/content/users-guide.

———. 2013b. *The ROOT Reference Guide*. http://root.cern.ch/drupal/content/reference-guide.