MpddstSelector

Kacper Skelnik, Faculty of Physics, Warsaw University of Technology
contact: kacske@wp.pl

August 27, 2019

Contents

1 Introduction

The following instructions describe programs written to do physical analysis using the PROOF system. It also discusses the structure of these programs, so that the reader will be able to easily modify them or write their variation of this programs. The system I propose contains the files: " MpddstSelector.C ", " MpddstSelector.h ", " ReadAllRootFiles.C ", " runAnalyze.sh ".

2 PROOF

PROOF is a tool that is part of the ROOT environment, enabling parallel processing of data on many processor cores. More specifically, PROOF splits the user-defined work into many independent subtasks. This is the opposite of batch processing. Unfortunately, macro which PROOF can perform should be written in a specific way. The following chapters describe the MpddstSelector - macro written this way.

To start PROOF, enter the following command in the ROOT program:

```
root [0] TProof::Open("")
 +++ Starting PROOF-Lite with 8 workers +++
Opening connections to workers: OK (8 workers)
Setting up worker servers: OK (8 workers)
PROOF set to parallel mode (8 workers)
(TProof *) 0x56173fce9970
```

```
root [1]
```

We can modify, e.g. the number of cores that our analysis should use. We do this by adding:

```
root [1] TProof::Open("","workers=2")
 +++ Starting PROOF-Lite with 2 workers +++
Opening connections to workers: OK (2 workers)
Setting up worker servers: OK (2 workers)
PROOF set to parallel mode (2 workers)
(TProof *) 0x56173fce9970
root [2]
```

More information at https://root.cern.ch/starting-proof

3 Selector

The selector is a program necessary to use the PROOF system. It contains two files: Selector.C and Selector.h.TSelector is a class that is part of the ROOT environment that allows us to use functions that PROOF knows how to perform. The outline of the header file should look like this:

```
#ifndef MpddstSelector_h
#define MpddstSelector_h

#include <TROOT.h>
#include <TChain.h>
#include <TFile.h>
#include <TSelector.h>
#include <TTreeReader.h>
#include <TTreeReaderValue.h>

class MpddstSelector : public TSelector {
public :
        TTreeReader     fReader;
        TTree          *fChain = 0; // declaration of the pointer for TTree or TChain analysis

        TH1D *hpt; // Deklaracja histogramow

        TTreeReaderValue<MpdEvent> MPDEvent_; // TTree branch pointer declaration

        TFile *opfile; // declaration of the output file

        Dwie_galezie(TTree * /*tree*/ =0) MPDEvent_(fReader, "MPDEvent."),
        hpt(0) { } // default constructor, remember to define all objects
        virtual ~Dwie_galezie() { }
        virtual Int_t   Version() const { return 2; }
        virtual void    Begin(TTree *tree);
        virtual void    SlaveBegin(TTree *tree);
        virtual void    Init(TTree *tree);
        virtual Bool_t  Notify();
        virtual Bool_t  Process(Long64_t entry);
        virtual Int_t   GetEntry(Long64_t entry, Int_t getall = 0) { return fChain ?
```

```
        fChain->GetTree()->GetEntry(entry, getall) : 0; }
        virtual void    SetOption(const char *option) { fOption = option; }
        virtual void    SetObject(TObject *obj) { fObject = obj; }
        virtual void    SetInputList(TList *input) { fInput = input; }
        virtual TList  *GetOutputList() const { return fOutput; }
        virtual void    SlaveTerminate();
        virtual void    Terminate();

        ClassDef(MpddstSelector,0);
};
#endif

#ifdef MpddstSelector_cxx
void MpddstSelector::Init(TTree *tree){
        mpdloadlibs(); // Loading libraries
        if (!tree) return;
                fChain = tree;
        fReader.SetTree(tree); // Assign the appropriate TTree object to the pointer
    }
Bool_t MpddstSelector::Notify(){
        return kTRUE; // you can retrieve branch pointers here
    }
#endif
```

4 Description of TSelectos functions

Description of the functions you need for Selector's operation can be found e.g. under links:

- `https://root.cern.ch/developing-tselector`
- `https://root.cern.ch/processing-proof`

There are also examples of programs written using PROOF, and guides. Below I will describe the most important function again:

- Begin() - I call at the beginning of the macro. You can define here, for example an output file.
- SlaveBegin() - The function called after the Begin() function for each "worker". It is place for defined hithistograms, and added their to the OutputList using the GetOutputList() function.
- Process() - A function that is called every time we download data from the TTree for processing data. It's here there should be the body of the analysis. The function returns a logical value that should not be used anywhere (kTRUE by default).
- Terminate() - Function called last. Most often used to save ready data to a file, and presenting them on the histograms.
- SlaveTerminate() - A function called at the end of the work of each "worker"
- Init() - Function called when selector initializing a new TTree or network TChain. You can load libraries here, check the existence of the tree, count the number of events. Remember to declare a tree here for TTreeReader.
- Notify() - Function called when a new file is received.

5 Secletor.C

The source code file should contain the function responsible for analyzing the data and creating histograms:

```
#define MpddstSelector_cxx

#include "MpddstSelector.h"
#include <TH1.h>
#include <TStyle.h>

void MpddstSelector::Begin(TTree * /*tree*/){
    TString option = GetOption();

    opfile = new TFile("file.root", "RECREATE"); // definition of the output file
}
void MpddstSelector::SlaveBegin(TTree * /*tree*/){
    TString option = GetOption();

    hpt = new TH1D(..definiowanie histogramu..);
    hpt->Sumw2(); // adding errors
    GetOutputList()->Add(hpt); // add the histogram to the list of output files
}
Bool_t MpddstSelector::Process(Long64_t entry){
    fReader.SetLocalEntry(entry);// specify the input number (replaces "event loop")
    /* analysis */

    return kTRUE;
}
void MpddstSelector::Terminate(){
    opfile->cd();
    hpt->Write(); // save the histogram to a file
}
```

6 Loading data from root files

To load the root file into TChain, run root and enter the following in the console:

```
root[0] TChain* myChain = new TChain("cbmsim")
root[1] myChain->AddFile("plik.root")
```

However, we are interested in the possibility of analyzing many root files at once. In that case, it's easy to say that the above methodloading files is inefficient. The "ReadAllRootFiles.C" macro is used to automate this process. According from the needs, you can manually define the range of folders read by the macro (first way to load root files):

```
for(Int_t iter=BEGIN; iter<=END; ++iter){
```

or use a text file with (secound way to load root files):

```
ifstream *istr = new ifstream("./urqmd34-11gev.list.txt");
```

To load the macro, type in the console:

```
root[1] .L ReadAllRootFiles.C
root[2] ReadAllRootFiles(myChain)
```

7 Starting the data analysis process

In summary, all the commands that you need to enter in the console to run physical analysis are:

```
root -l -b
root [0] TProof::Open("")
root [1] TChain* myChain = new TChain("cbmsim")
root [2] .L ReadAllRootFiles.C
root [3] ReadAllRootFiles(myChain)
root [4] myChain->SetProof()
root [5] myChain->Process("MpddstSelectorOla.C")
```

To optimize the process fot the users, the script "runAnalyze.sh" was written. It it allows to run analysis by just enter in the console:

```
./runAnalyze.sh
```