PROOF Basics

ROOT Tutorials

2013
Open a PROOF-Lite session

• To start a PROOF-Lite session

```bash
$ root -l
root [0] TProof::Open(""")
  +++ 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)
(class TProof*)0x8330140
root [1] gProof
(class TProof*)0x8330140
```

• Now we are ready to go
• gProof is a global instance of TProof
• … but what’s TProof?
TProof: the PROOF shell

- TProof is the interface class to interact with the PROOF session
- Everything in the session is done via TProof
  - `Print()`, gives information about the session
  - `Process()`, allows to run a TSelector
  - `GetOutputList()`, returns the list of output objects
  - `DrawSelect()`, draws distributions
  - ...
- … and many others. Check:

**TProof::Print()**

- Gives information about the session

```
$ root [1] gProof->Print()
*** PROOF-Lite cluster (parallel mode, 2 workers):
Host name:    macphsft12.local
User:         ganis
ROOT version|rev|tag:     5.32/02|r43514
Architecture-Compiler: macrosx64-gcc421
Protocol version:    33
Working directory:  /Users/ganis/local/root/opt/root
Communication path: /var/folders/uc/uc0rgjquflmzr689bg+jju++0gq/-tmp-/plite-38583
Log level:       0
Number of workers: 2
Number of active workers: 2
Number of unique workers: 1
Number of inactive workers: 0
Number of bad workers: 0
Total MB's processed: 0.00
Total real time used (s): 0.000
Total CPU time used (s): 0.000
```
Ordinal numbers

• Unique identifier
  – 0.n for workers (master is always 0)

• The sandbox
  – Location controlled by Proof.Sandbox
    Proof.Sandbox /my/special/sandbox
  – Defaults
    • $HOME/.proof (standard PROOF)
    • $HOME/.proof/path-where-we-started (PROOF-Lite)
Sandbox content

- **cache**
  - for package tarballs, code, binaries
- **packages**
  - where packages are actually build
- **queries** (master or PROOF-Lite only)
  - where the results from queries are stored
- **datasets** (master or PROOF-Lite only)
  - where information about datasets is stored
- **session-SessionUniqueID**
  - Working area (logs, …) for session SessionUniqueID
First processing (first exercise)

• The file **ProofSimple.C.h** define a TSelector which fills 100 histograms with gaussian random numbers

```c
root[] gProof->SetParameter("ProofSimple_Nhist", (Long_t)100)
root[] gProof->Process("ProofSimple.C+",10000)
```
The same without PROOF

• Repeat on the local session

```c
root[] TProofPlayerLocal *p = new TProofPlayerLocal();
root[] p->AddInput( \n    new TParameter<Long_t>("ProofSimple_Nhist", 100)
root[] p->Process("ProofSimple.C+",10000)
```

• Try to measure the time with gROOT->Time()
  – What do you find?
TProofBench: another exercise

• **TProofBench** is a tool to benchmark a PROOF cluster (lite or full installation)

• Framework for CPU-intensive and I/O-intensive scalability tests

• Default benchmarks
  – CPU-intensive
    • Random number generation
  – I/O-intensive
    • Readout TTrees based on $ROOTSYS/test/Event.C,.h

• **Exercise 14b**
The dialog box

Selector being run

Active workers

Progress bar

Stats

Log dialog box
The log dialog box

Select logs to display

Can be started also with TProof::fLogViewer

Grep functionality
Save to a file
Where are the results?

- In the output list

```
root [] gProof->GetOutputList()
(class Tlist*)0x89eae58
root [] gProof->GetOutputList()->ls()
OBJ: TStatus PROOF_Status : 0 at: 0x8a264a8
OBJ: TH1F h0 h0 : 0 at: 0x89d5b48
OBJ: TH1F h1 h1 : 0 at: 0x8a22de0
OBJ: TH1F h2 h2 : 0 at: 0x8a21f88
OBJ: TH1F h3 h3 : 0 at: 0x8a215f8
OBJ: TH1F h4 h4 : 0 at: 0x8a24100
OBJ: TH1F h5 h5 : 0 at: 0x8a288b8
OBJ: TH1F h6 h6 : 0 at: 0x8a31c20
...
OBJ: TH1F h96 h96 : 0 at: 0x89e9b38
OBJ: TH1F h97 h97 : 0 at: 0x89ea000
OBJ: TH1F h98 h98 : 0 at: 0x89ea4c8
OBJ: TH1F h99 h99 : 0 at: 0x89ea990
root []
```
Processing data: EventData example

• Generate a few files, for example 10

```c
root [] .L CreateEventTree.C
root [] Int_t i = 0;
root [] for(;i<10;++i){
    CreateEventTree(Form("data/evtree_%d",i));}
```

• Create a chain

```c
root [] TChain c("tree")
root [] for(i=0;i<10;++i){
    c.AddFile(Form("file:///home/user/data/evtree_%d",i));}
```

• Process the chain inside PROOF

```c
root [] c.SetProof()
root [] c.Process("EventDataSelector.C+")
```
Processing data: EventData example

- Use gROOT->Time() to measure **speed-up**
  - No PROOF

```
root [] c.SetProof(0)
root [] gROOT->Time();
root [] c.Process("EventDataSelector.C+")
```

- PROOF

```
root [] c.SetProof()
root [] gROOT->Time();
root [] c.Process("EventDataSelector.C+")
```
The concept of dataset is very useful in HEP: it refers to a set of files containing homogeneous data.
- e.g. all the data taken during Summer 2009 under uniform detector conditions.

Is useful to refer to a dataset by name

TFileCollection: named list of TFileInfo

TFileInfo: most generic way of describing a file
- Multiple URLs, meta-information

A TFileCollection is the typically the result of a query to a catalog
Datasets in PROOF

- TProof has a set of methods to perform basic operations on datasets
  - `RegisterDataSet` (const char *`name`, TFileCollection *)
  - `VerifyDataSet` (const char *`name`)
  - `ShowDataSets`()
  - TFileCollection *`GetDataSet` (const char *`name`)
  - ...

- The name is in the form `/group/user/datasetname`
  - 'group' is an advanced PROOF concept: by default anybody is in group 'default'
Register / verify a dataset

- Second part of **Exercise 14c**
- Create a dataset for the files we used in the previous example
- Register and Verify it
- Process the dataset ‘by name’
Loading additional code

• When the selector needs additional code - for example a new class defined in the files MyClass.C and MyClass.h – PROOF provides two ways to make it available
  
  – `gProof->Load("MyClass.C")`
    • Equivalent of `.L` on the ROOT shell
    • Convenient for simple things
  
  – **Package A**Rchives (PAR)
    • Structured archives with build and setup facilities
    • Convenient for more complex and stable things, e.g. the experiment analysis suite
PARs

- Zipped **tarballs** identified by a name and the .par extension, e.g. `pack.par`
- The tarball contains a structure like this
  ```
  ./pack
  ./pack/PROOF-INF
  ./pack/PROOF-INF/BUILD.sh
  ./pack/PROOF-INF/SETUP.C
  ```
- The code (.C, .h, makefiles, ...) should be put in the top level directory
- **BUILD.sh**: script to build the package, e.g. runs 'make'
- **SETUP.C**: is a macro running the final setup
Pythia8 example

• This is an example of running a real Monte Carlo simulation in PROOF
  – Pythia8 is the first usable C++ version of a famous HEP generator
• It needs a PAR file to setup the environment
• To run the example we need to link to pythia8
  – Available under /cvmfs/sft.cern.ch/lcg/dev
```c
void SETUP()
{
    // Load the libraries
    gSystem->Load("$PYTHIA8/lib/libpythia8.so");
    gSystem->Load("libEG");
    gSystem->Load("libEGPythia8");

    // Set the include paths
    gROOT->ProcessLine(".include $PYTHIA8/include");
}
```
Pythia8 exercise

- Exercise 14d
- Download the pythia8.par and the related selectors
- Run the code to produce the following plot
Summary

• We have seen the basic concepts of PROOF
  – How to run a selector for a CPU intensive job
  – How to process a TChain
  – How to do the same from a named dataset
  – How to use a PAR file