PROOF integration in FAIRROOT

Radoslaw Karabowicz
GSI

12.12.2011
XXIX PANDA Collaboration Meeting

What is PROOF?

GridKa 2011, ROOT and PROOF Tutorial

PROOF stands for Parallel ROOt Facility.

It allows parallel processing of large amount of data. The output results can be directly visualized (e.g. the output histogram can be drawn at the end of the proof session).

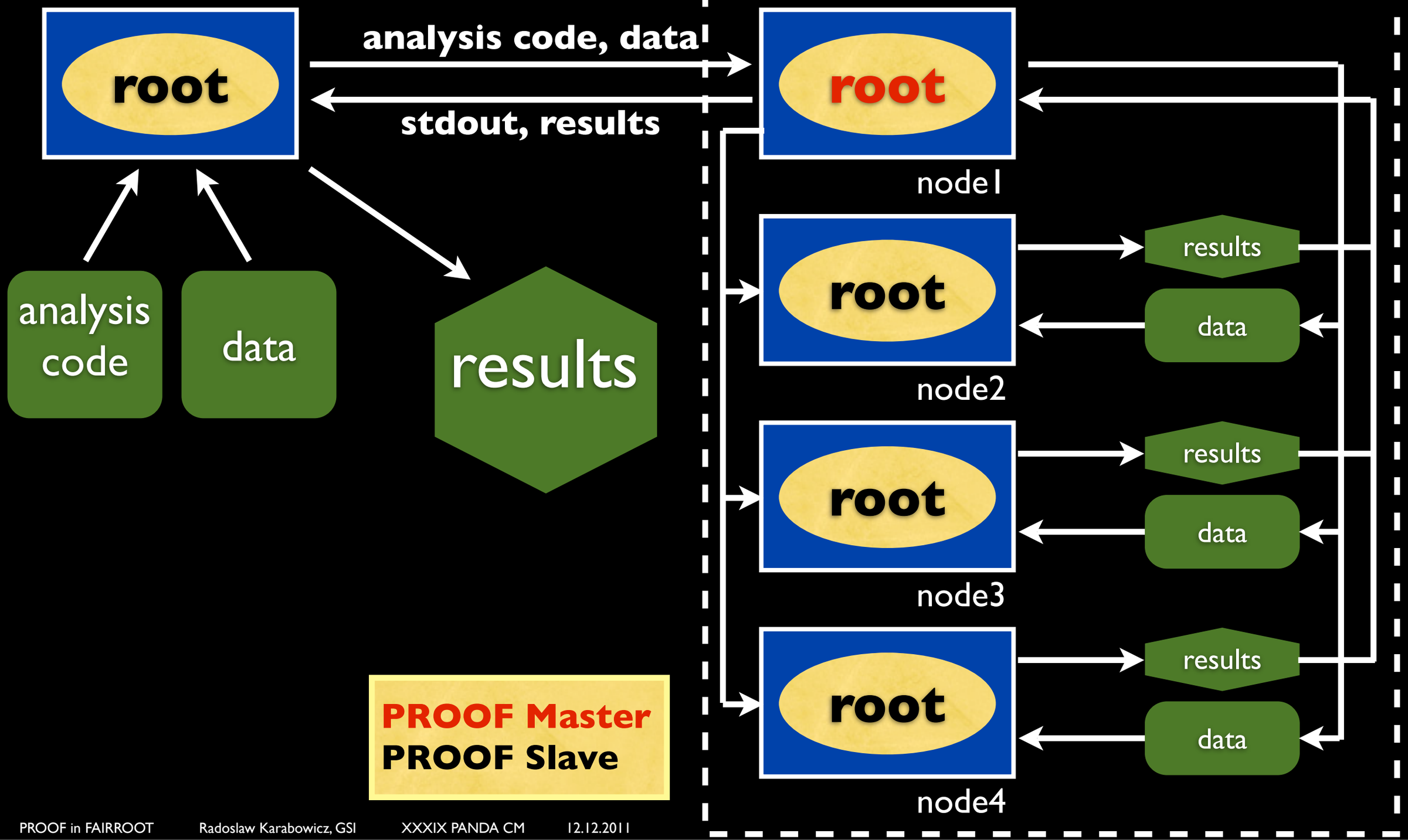
PROOF is NOT a batch system.

The data which you process with PROOF can reside on your computer, PROOF cluster disks or grid.

The usage of PROOF is transparent: you should not rewrite your code you are running locally on your computer.

No special installation of PROOF software is necessary to execute your code: PROOF is included in ROOT distribution.

How does PROOF work?

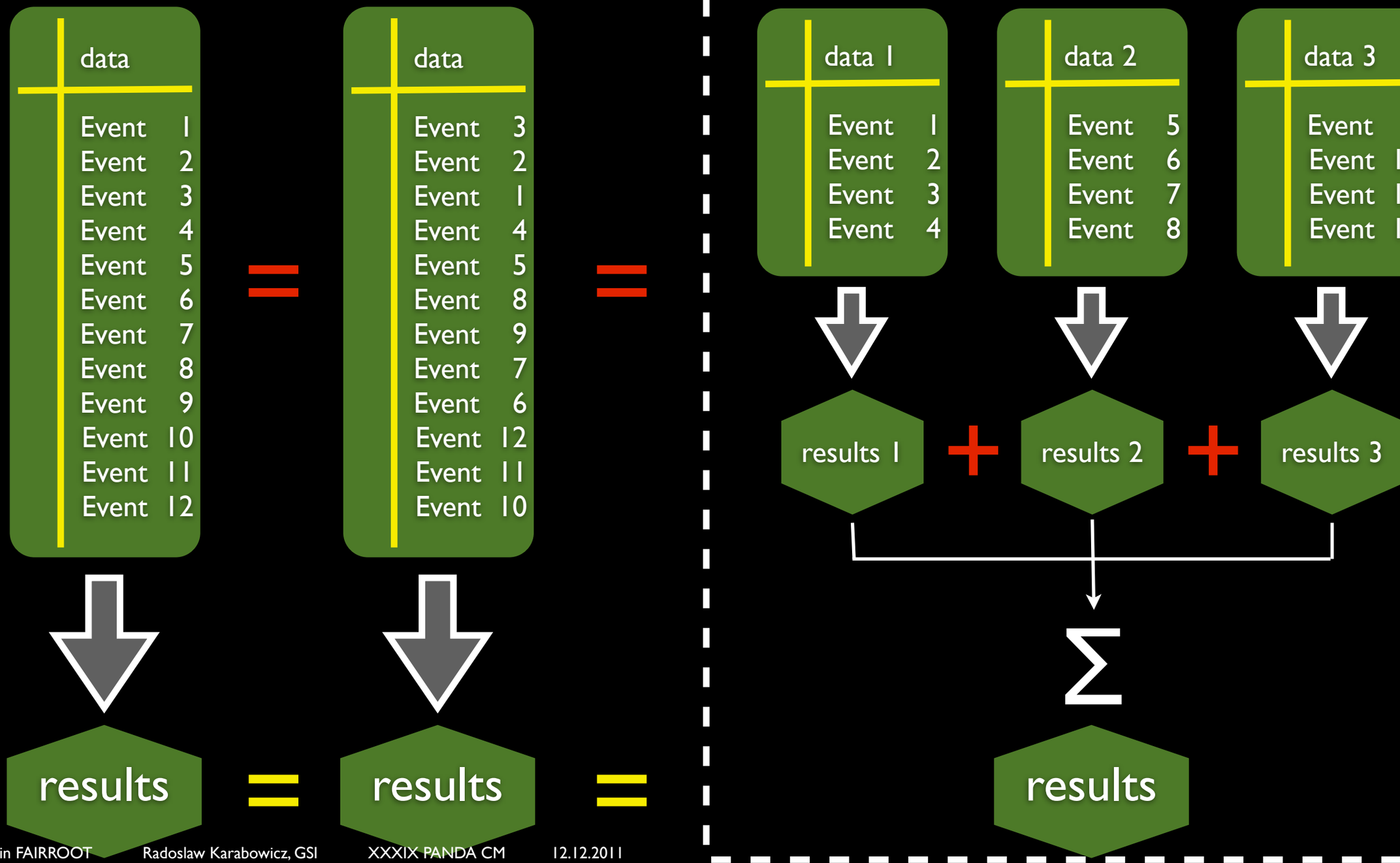


Trivial parallelism

Sequential processing

Unordered processing

Parallel processing



PROOF terminology

The following terms are used in PROOF:

PROOF cluster
Set of machines communicating with PROOF protocol. One of those machines is normally designated as Master (multi-Master setup is possible as well). The rest of machines are Workers.

Client
Your machine running a ROOT session that is connected to a PROOF master.

Master
Dedicated node in PROOF cluster that is in charge of assigning workers the chunks of data to be processed, collecting and merging the output and sending it to the Client.

Slave/Worker
A node in PROOF cluster that processes data.

Query
A job submitted from the Client to the PROOF cluster.
A query consists of a selector and a chain.

Selector
A class containing the analysis code

Chain
A list of files (trees) to process

PROOF Archive (PAR) file
Archive file containing files for building and setting up a package on the PROOF cluster. Normally is used to supply extra packages used by user job.

PROOF-Lite
PROOF cluster that uses only locally available CPU cores

PROOF basics

- Easy to start (24 keyboard strokes):

```
karabowi@kp3mac001::~$ 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*)0x10187fc00
root [1]
```

- Easy to use (process selector on a chain):

```
root [1] TChain* myChain = new TChain("cbmsim")
root [2] myChain->AddFile("myFile.root")
(Int_t)1
root [3] myChain->SetProof()
root [4] myChain->Process("MySelector.C")
```

PROOF basics

- User needs to develop a selector:

```
karabowi@kp3mac001::~$ root -l
root [0] TChain* myChain = new TChain("cbmsim")
root [1] myChain->AddFile("myFile.root")
(Int_t)1
root [2] myChain->MakeSelector("MySelector")
Warning in <TClass::TClass>: no dictionary for class PndMCTrack is available
Warning in <TClass::TClass>: no dictionary for class PndSdsMCPoint is available
Warning in <TClass::TClass>: no dictionary for class FairMCPoint is available
Warning in <TClass::TClass>: no dictionary for class FairBasePoint is available
Warning in <TClass::TClass>: no dictionary for class FairTimeStamp is available
Warning in <TClass::TClass>: no dictionary for class FairMultiLinkedData is available
Warning in <TClass::TClass>: no dictionary for class FairLinkedData is available
Warning in <TClass::TClass>: no dictionary for class PndSttPoint is available
Warning in <TClass::TClass>: no dictionary for class PndGemMCPoint is available
Warning in <TClass::TClass>: no dictionary for class PndTofPoint is available
Warning in <TClass::TClass>: no dictionary for class FairMCEventHeader is available
Warning in <TClass::TClass>: no dictionary for class FairFileHeader is available
Info in <TTreePlayer::MakeClass>: Files: MySelector.h and MySelector.C generated from
TTree: cbmsim
(Int_t)0
root [4]
```

PROOF basics

- MySelector.h contains full list of the TTree branches, and few functions that can be filled by the user:

```
virtual void    Begin(TTree *tree);         // executed on master at the beginning
virtual void    SlaveBegin(TTree *tree);    // executed on each worker node at the beginning
virtual void    Init(TTree *tree);          // executed on a worker when getting new tree
   virtual Bool_t  Notify();
   virtual Bool_t  Process(Long64_t entry);  // executed for event "entry" in the tree
   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();           // executed on each worker node at the end
   virtual void    Terminate();                // executed on master at the end
```

- Input and output controlled via TLists:

```
   TList* fInput;          // list of objects available during processing
   TSelectorList* fOutput // list of objects created during processing
```

PROOF in FAIRROOT

GOALS:

- run FAIRROOT analysis on PROOF cluster
- restrict the changes to fairbase, i.e.
- reduce the changes in users' analysis code &
- reduce the changes in users' macros

PROOF in FAIRROOT

STEPS:

- load FAIRROOT libraries on the workers
- develop general selector
- change fairbase *et al*

PROOF Archive (PAR)

- gtar'red directory containing SETUP.C and optionally BUILD.sh. These scripts will be executed on each worker node
- GOAL: have to load FAIRROOT libraries on each worker node

- IMPORTANT: need a simple way to get list of needed libraries; this solution has to be general for each experiment using FAIRROOT and has to require minimum users' intervention
- SOLUTION: current implementation of libFairRoot.par contains only SETUP.C which loads and executes gconfig/rootlogon.C

FairAnaSelector

- The class deriving from TSelector with well defined member functions that are executed in specific order. Usually used as `myChain->Process(MySelector);` either locally or on PROOF
- GOAL: send a FairRunAna with the list of tasks, parameters, geometry, etc. to the workers, analyze the chain, collect workers' outputs and merge the outputs
- CHALLENGES: to send objects to the workers via TList* fInputList the objects have to be "streamable"

FairAnaSelector

- “Streamable”? = ~simple~
  - no instantons
  - derive MyClass from TObject
  - public default constructor MyClass();
  - initialize all members to 0

FairAnaSelector

- SOLUTION: master FairRunAna opens proof session, adds outputFileName, parameterFileNames, fTask to fInput, uploads .par package and runs FairAnaSelector on the input chain:

```
TProof* proof = TProof::Open(fProofServerName.Data());

proof->AddInput(new TNamed("FAIRRUNANA_fOutputFileName",outFile.Data()));
proof->AddInput(new TNamed("FAIRRUNANA_fParInput1FName",par1File.Data()));
proof->AddInput(new TNamed("FAIRRUNANA_fParInput2FName",par2File.Data()));
proof->AddInput(fTask);

proof->UploadPackage(fProofParName.Data());
proof->EnablePackage(fProofParName.Data());

inChain->SetProof();
inChain->Process("FairAnaSelector","",NEntries,NStart);
```

FairAnaSelector

- SOLUTION: FairAnaSelector creates FairRunAna on each worker node at the begin of the job, it asks this FairRunAna to analyze individual events in the Process() function, the FairRunAna is finished function and its output is stored in TSelectorList* fOutput in the SlaveTerminate() function.
- OPTIONAL: The default ROOT's file/tree merger may be used to merge the workers' output. It is also possible to store the individual workers' outputs.

PROOF in FAIRROOT

```
myMacro{ //running locally
  FairRunAna* fRun;
  fRun->SetInputFile();
  fRun->SetOutputFile();
  FairRuntimeDb* rtdb =
    fRun->GetRuntimeDb();
  rtdb->SetFirstInput();
  rtdb->SetSecondInput();

  fRun->AddTasks();
  fRun->Init();
  fRun->Run(firstEvent,
            lastEvent);
}
```

```
void FairRunAna::Run
          (Int_t NStart,
           Int_t NStop) {
  for(Int_t iev=NStart;
            iev<NStop;
            iev++) {
    fTask->ExecuteTask()
  }
}
```

```
myMacro {  // running on PROOF
  FairRunAna* fRun;
  fRun->SetInputFile();
  fRun->SetOutputFile();
  FairRuntimeDb* rtdb = fRun->GetRuntimeDb();

  rtdb->SetFirstInput();
  rtdb->SetSecondInput();

  fRun->AddTasks();
  fRun->Init();
  fRun->Run(firstEvent, lastEvent,"proof");

}
```

```
void FairRunAna::Run
          (Int_t NStart,
              Int_t NStop, const char* type) {
   TProof* proof = TProof::Open("");
   proof->AddInput(outFileName.Data());
   proof->AddInput(par1FileName.Data()));
   proof->AddInput(par2FileName.Data()));
   proof->AddInput(fTask);
   proof->UploadPackage("libFairRoot.par");
   proof->EnablePackage("libFairRoot.par");
   inChain->SetProof();
   inChain->Process("FairAnaSelector",
                    "",NEntries,NStart);
}
```

```
FairAnaSelector::Init(TTree* tree) {
 if ( !fRunAna ) {
  fRunAna = new FairRunAna();
  fRun->SetInTree(tree);
  fRun->SetOutputFile(
      fInput->FindObject(outFileName));

  FairRuntimeDb* rtdb =
        fRun->GetRuntimeDb();
  rtdb->SetFirstInput(
      fInput->FindObject(par1FileName));
  rtdb->SetSecondInput(
      fInput->FindObject(par2FileName));

  fRun->AddTasks
   (fInput->FindObject("FairTaskList"));

  fRun->Init();
 }
 else {
  fRunAna->SetInTree(tree);

  FairRootManager* ioman =
    FairRootManager::Instance();
  ioman->OpenInTree();
 }
}

FairAnaSelector::Process(Long64_t entry){
  fRunAna->RunEntry(entry);
}
```

fairbase *et al*

- On these slides several most important currently implemented changes to FAIRROOT will be summarized:

```
karabowi@lxi012::~/pandaroot_13510/trunk/base$ svn status
?       FairAnaSelector.cxx
?       FairAnaSelector.h
M       FairRun.cxx
M       FairRootManager.cxx
M       FairRun.h
M       FairTask.cxx
M       FairRootManager.h
M       CMakeLists.txt
M       FairRunInfo.cxx
M       FairRunAna.cxx
M       FairTask.h
M       FairLinkDef.h
M       FairRunAna.h
karabowi@lxi012::~/pandaroot_13510/trunk/base$ svn diff | wc -l
    1714
karabowi@lxi012::~/pandaroot_13510/trunk/base$ svn status
M       FairParRootFileIo.cxx
M       FairParAsciiFileIo.cxx
M       FairParIo.h
M       FairParAsciiFileIo.h
M       FairParIo.cxx
M       FairDetParRootFileIo.cxx
karabowi@lxi012::~/pandaroot_13510/trunk/base$ svn diff | wc -l
     101
```

```
karabowi@.../trunk/parbase$ cd ../gem
karabowi@.../trunk/parbase/gem$ svn diff | wc -l
     349
karabowi@.../trunk/parbase/gem$ cd ../mvd
karabowi@.../trunk/parbase/mvd$ svn diff | wc -l
     231
karabowi@.../trunk/parbase/mvd$ cd ../sds
karabowi@.../trunk/parbase/sds$ svn diff | wc -l
     736
karabowi@.../trunk/parbase/sds$ cd ../stt
karabowi@.../trunk/parbase/stt$ svn diff | wc -l
      81
karabowi@.../trunk/parbase/stt$ cd ../global
karabowi@.../trunk/parbase/global$ svn diff | wc -l
     407
```

fairbase *et al*

FairRunAna (only most important mentioned):

- new member:
    - TSelector* fSelector;

- new functions:
    - void Run(Int_t NStart =0,Int_t NStop=0, const char *type);
    - void RunEntry(Int_t entryNo);
    - void SetInChain(TChain* tempChain);
    - void SetInTree (TTree* tempTree);
    - TTree* GetOutTree();

fairbase *et al*

FairRootManager (only most important mentioned):

- new member:
    - TTree* fInTree;

- new functions:
    - void SetInTree (TTree* tempTree);
    - void SetInChain(TChain* tempChain);
    - Bool_t OpenInTree();
    - TObject* GetObjectFromInTree(const char* BrName);
    - TObject* ActivateBranchInInTree(const char* BrName);

fairbase *et al*

MyTask (only most important mentioned):

- initialize all possible members to 0 in default constructor MyTask();
- initialize all possible members to 0 in default constructor MyClass() of class MyClass, which is a member of MyTask,
- pointers to instantons as members are difficult to stream
- do not ->Delete() empty pointers, protect with if: if (myPointer) myPointer->Delete();

Running PROOF

- The simplest way to use PROOF is the PROOF-Lite, which uses your own local machine CPUs:

```
karabowi@kp3mac001::~$ 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*)0x10187fc00
root [1]
```

- For creating a PROOF cluster that uses external CPUs one may use PoD (PROOF-on-Demand: http://pod.gsi.de)

PoD & PROOF

```
konglaide@kp3mac001:pod-server start
Starting PoD server...
updating xproofd configuration file...
starting xproofd...
starting PoD agent...
preparing PoD worker package...
selecting pre-compiled bins to be added to worker package...
PoD worker package will be repacked because "/Users/konglaide/.PoD/etc/xpd.cf" was
updated
PoD worker package: /Users/konglaide/.PoD/wrk/pod-worker
------------------------
XPROOFD [66174] port: 21001
PoD agent [66179] port: 22001
PROOF connection string: konglaide@kp3mac001.gsi.de:21001
------------------------
konglaide@kp3mac001::~$ pod-ssh -c ~/PoD/pod_ssh.cfg --submit
**   PoD jobs have been submitted. Use "pod-ssh --status" to check the status.
konglaide@kp3mac001::~$ pod-ssh -c ~/PoD/pod_ssh.cfg --status
PoD worker "etch64_16": RUN
PoD worker "etch64_21": RUN
PoD worker "etch64_20": RUN
konglaide@kp3mac001::~$ root -l
root [0] TProof::Open(gSystem->GetFromPipe("pod-info -c"))
Starting master: opening connection ...
Starting master: OK
Opening connections to workers: OK (20 workers)
Setting up worker servers: OK (20 workers)
PROOF set to parallel mode (20 workers)
(class TProof*)0x1018a9e00
root [1]
```

- Here a SSH plugin is used to connect to the workers
- Other plugins developed are: gLite, LSF, PBS, Grid Engine, Condor

Results

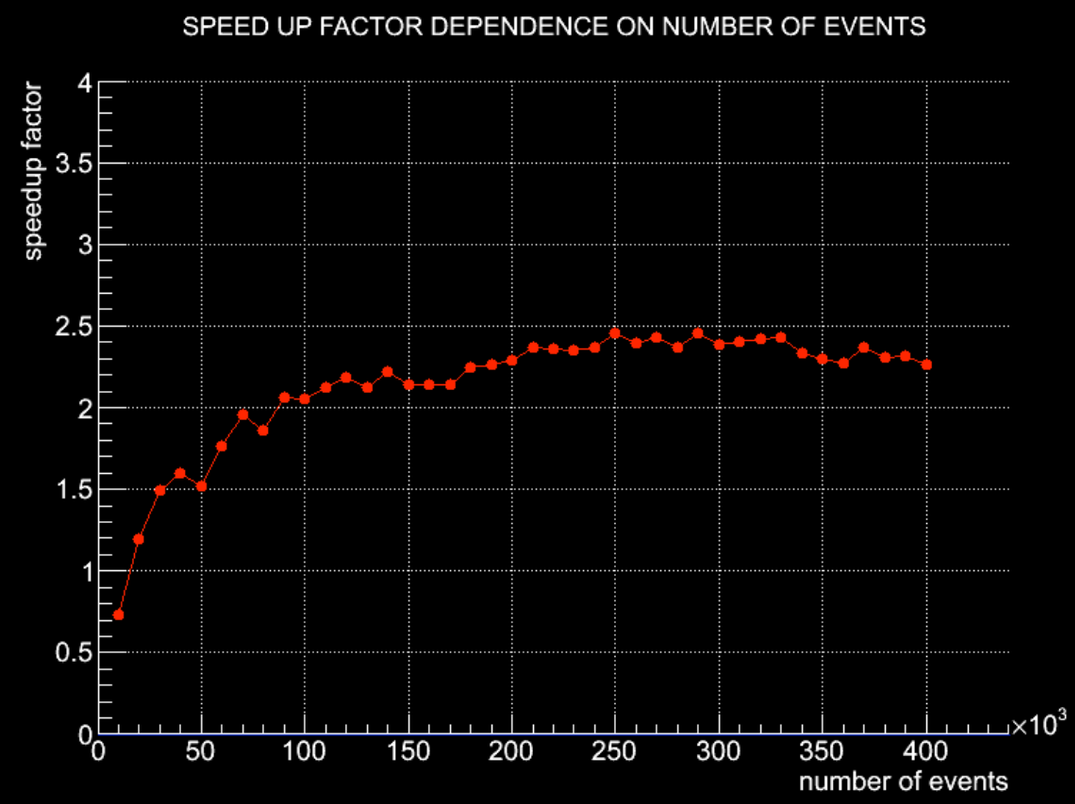
- Time performance
- Data quality
- Data integrity

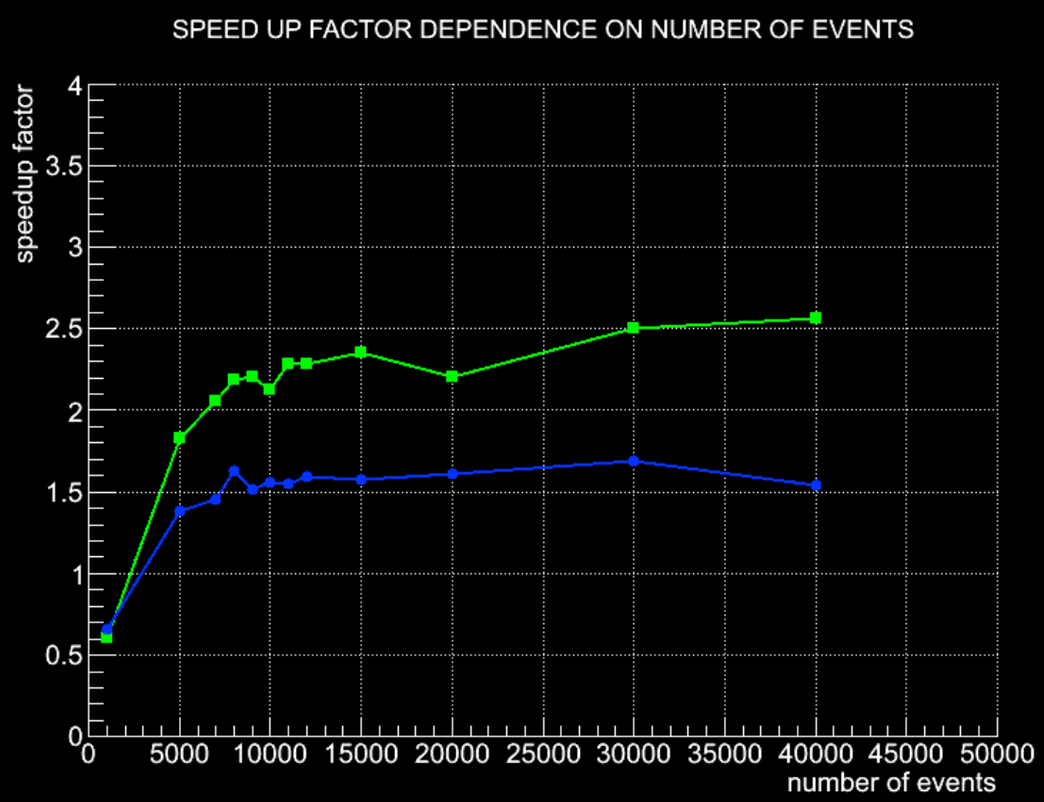
Time performance

- Some remarks:
- one will (almost) never get ideal scaling, so that n workers does not mean n time faster job execution, due to multiple initialization, library loading, PROOF overhead
- the IO limits the time performance

Time performance

- PROOF-Lite with 4 workers
- One task: PndGemFindHits
- speedup factor = local analysis time/proof analysis time



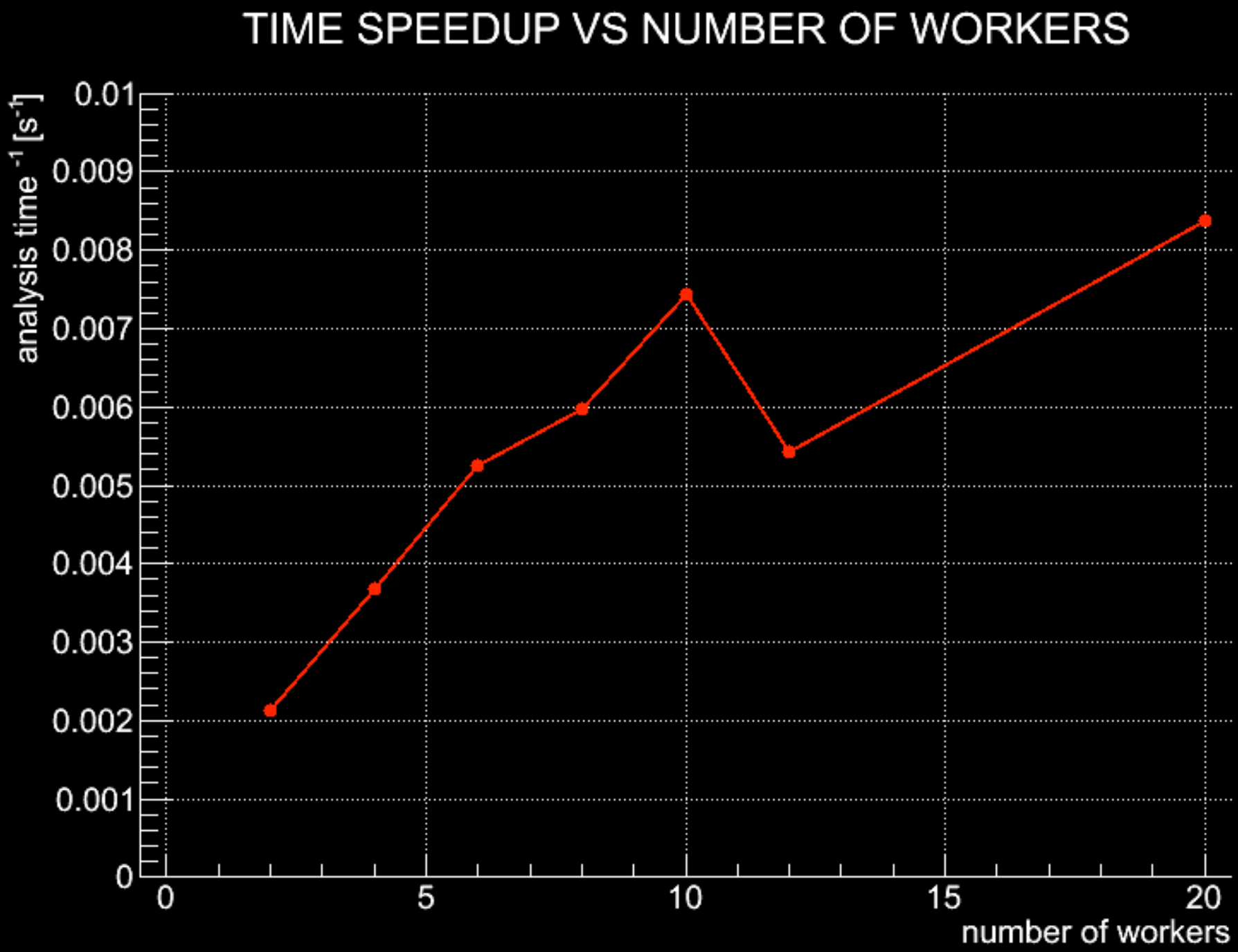


- Tasks: PndMvdDigiTask, PndMvdClusterTask, PndSttHitProducerIdeal, PndGemDigitize, PndGemFindHits, PndBarrelTrackFinder
- green: nWorkers = 4
- blue: nWorkers = 2

Time performance

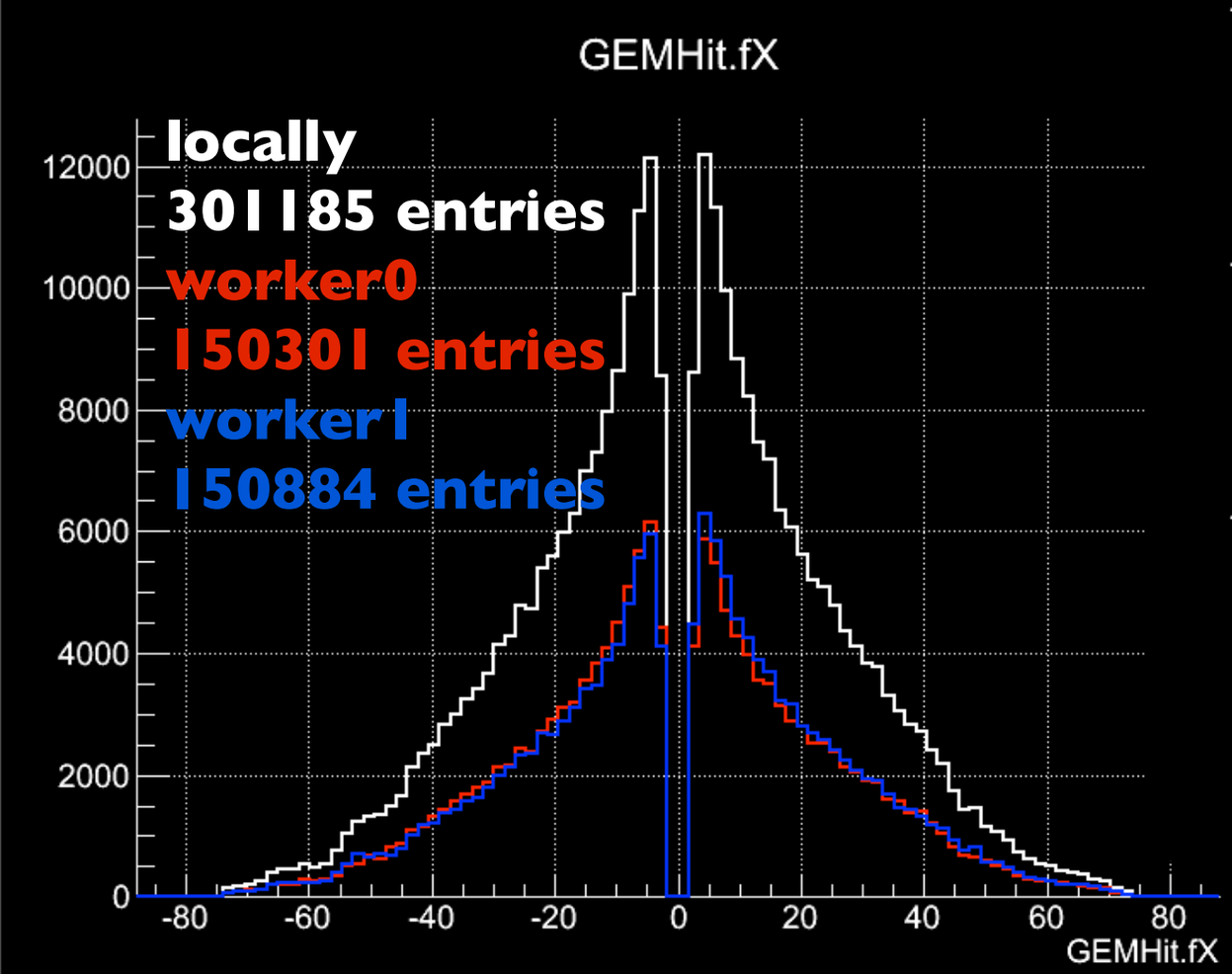
PROOF on external CPUs

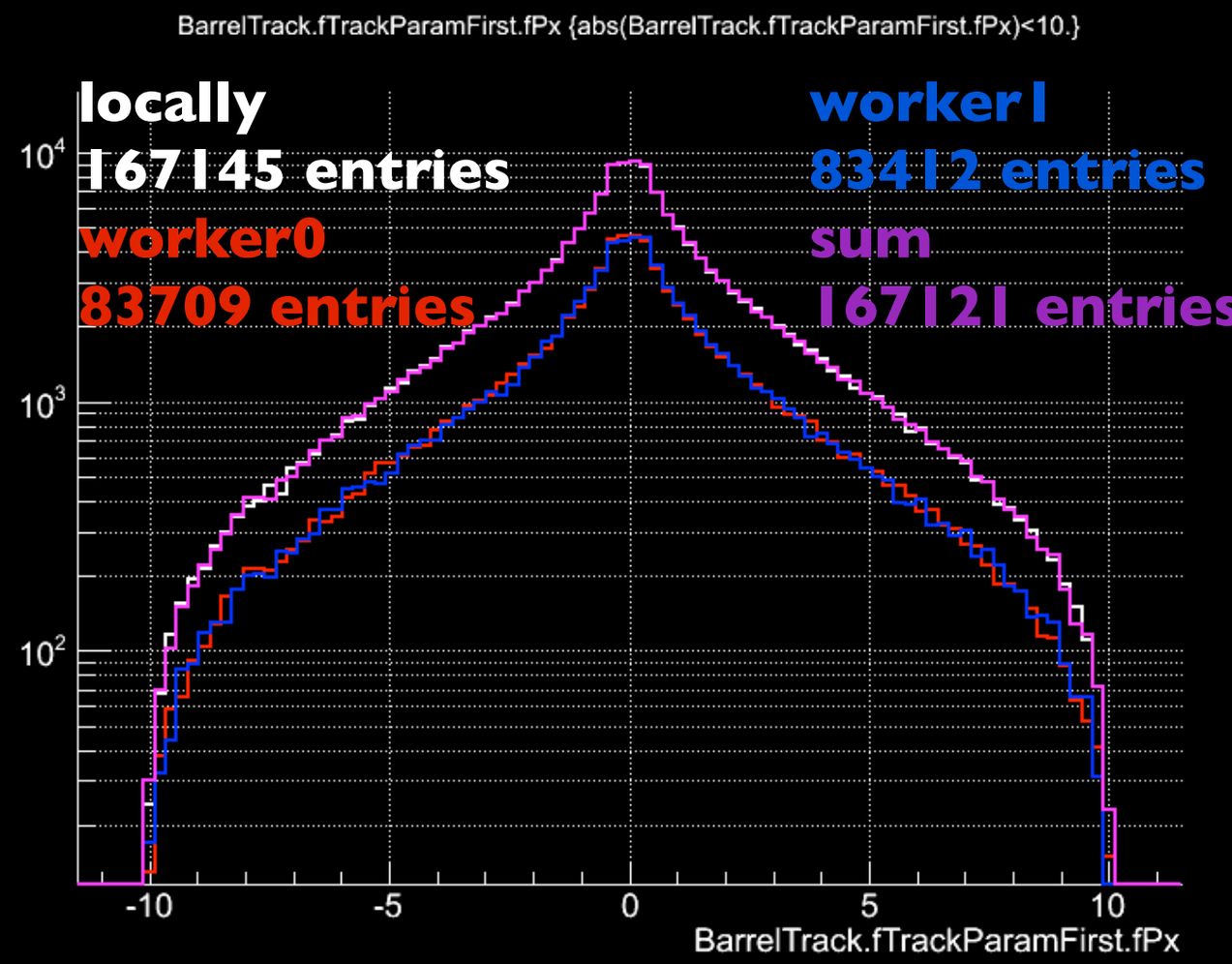
Using PoD with SSH plugin, lxi020 (4CPUs)+ lxi016 (8CPUs)+ lxi021 (8CPUs)



Data quality

- The result of the locally running FAIRROOT and the one running of PROOF are identical*



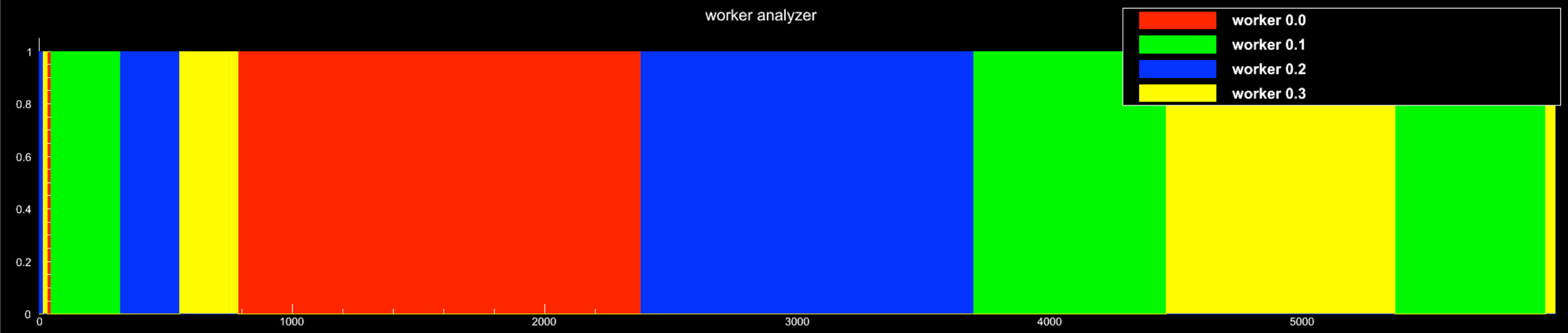


* - down to fRandom - the order of event processing is different locally and on PROOF

Data integrity

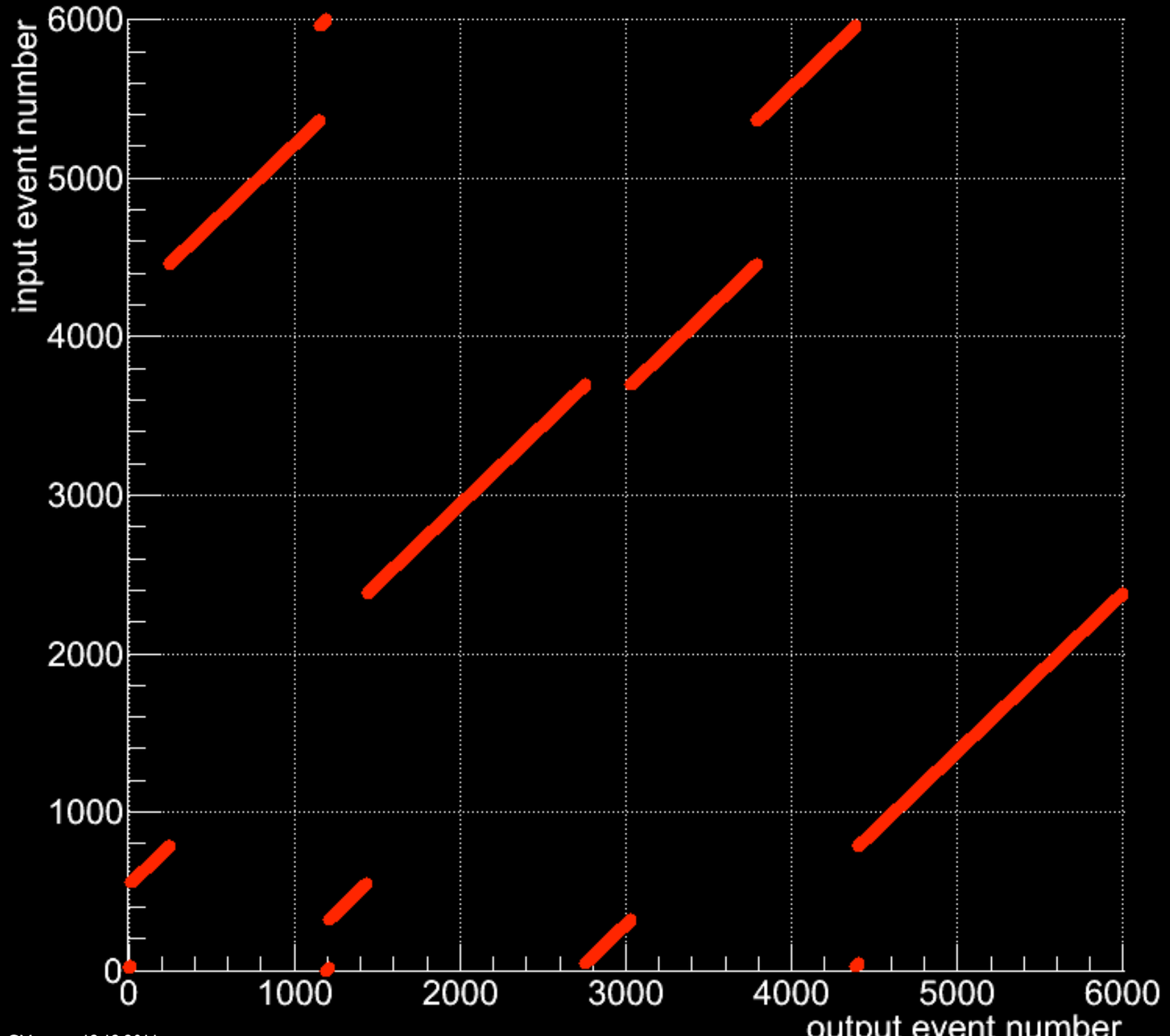
- The PROOF divides automatically the input data into chunks and distributes them among workers

- Extreme example: event distribution among PROOF workers:



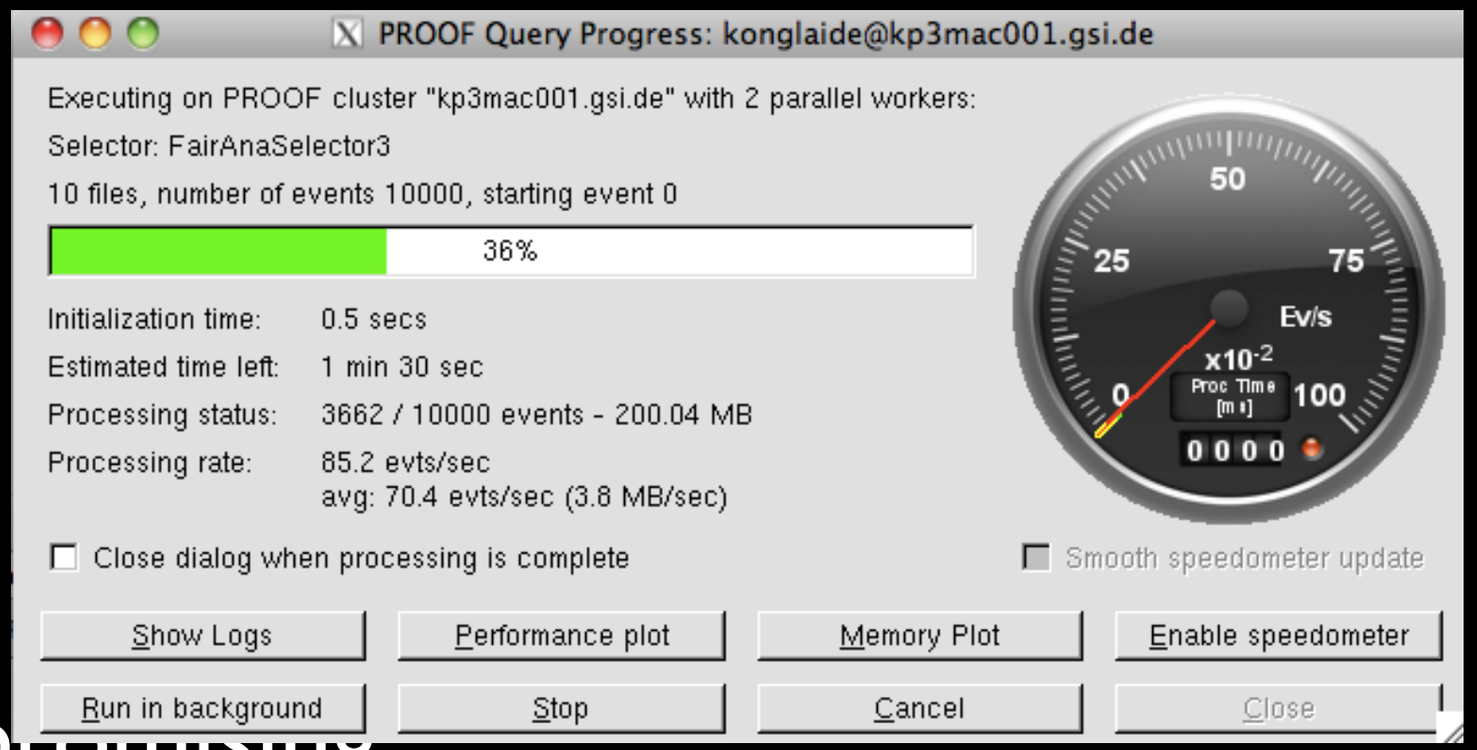
Data integrity

- Event order is mixed in the output file
- Extreme example: output vs input event order
- Extreme in a sense, that the mixing is on an event level. When more input files than worker nodes, the PROOF sends whole files to workers



Remarks

- FAIRROOT has been adopted to run on a PROOF cluster



- Tests results are promising
- Further work is still required
- The code is in the development branch and will be available in the trunk soon

Backup slides

Detailed processing time