Interactive Data Analysis on the Grid with PROOF and gLite

P.Malzacher@gsi.de

Anna Kreshuk, Peter Malzacher, Anar Manafov, Victor Penso, Carsten Preuss, Kilian Schwarz, Mykhaylo Zynovyev

International Symposium on Grid Computing
7-11 April 2008
Academia Sinica, Taipei, Taiwan
9 April 2008
GSI / FAIR

ALICE Computing

PROOF

PROOF on the Grid
GSI / FAIR

ALICE Computing

PROOF on the Grid

PROOF on the Grid
GSI - Gesellschaft für Schwerionenforschung
German National Centre for Heavy Ion Research

Budget: 95 Mio. € (90% Germany, 10% State of Hesse)

Employees: ~ 1000

External Scientific Users: 1000
Research Areas at GSI

**Nuclear Physics (50%)**
- Nuclear reactions up to highest energies
- Superheavy elements
- Hot dense nuclear matter

**Atomic Physics (15%)**
- Atomic Reactions
- Precision spectroscopy of highly charged ions

**Biophysics and radiation medicine (15%)**
- Radiobiological effect of ions
- Cancer therapy with ion beams

**Materials Research (5%)**
- Ion-Solid-Interactions
- Structuring of materials with ion beams

**Plasma Physics (5%)**
- Hot dense plasma
- Ion-plasma-interaction

**Accelerator Technology (10%)**
- Linear accelerator
- Synchrotrons and storage rings
### Added value
- Beam intensity by a factor of 100 - 10000
- Beam energy by a factor of 20
- Anti-matter beams and experiments
- Unique beam quality by beam cooling measures
- Parallel operation
- Data to be recorded in 2015: 1-10 times LHC

### Schedule, cost, user community
- Construction in three stages until 2015
- Construction cost: appr. 1 Billion Euro
- Scientific users: appr. 2500 - 3000 per year

### Funding (Construction)
- 65% Federal Republic
- 10% State of Hesse
- 25% International Partners
Plans for the Alice Tier 2&3 at GSI: Size

<table>
<thead>
<tr>
<th>Year</th>
<th>2007</th>
<th>2008</th>
<th>2009</th>
<th>2010</th>
<th>2011</th>
</tr>
</thead>
<tbody>
<tr>
<td>ramp-up</td>
<td>0.4</td>
<td>1.0</td>
<td>1.3</td>
<td>1.7</td>
<td>2.2</td>
</tr>
<tr>
<td>CPU (kSI2k)</td>
<td>400</td>
<td>1000</td>
<td>1300</td>
<td>1700</td>
<td>2200</td>
</tr>
<tr>
<td>Disk (TB)</td>
<td>120</td>
<td>300</td>
<td>390</td>
<td>510</td>
<td>660</td>
</tr>
<tr>
<td>WAN (Mb/s)</td>
<td>100</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>...</td>
</tr>
</tbody>
</table>

2/3 of that capacity is for the tier 2 (fixed via WLCG MoU)
1/3 for the tier 3

To support ALICE and to learn for FAIR computing.
GSI Setup: ~40% = ALICE Tier2/3 usable via batch, grid and PROOF

~1400 cores
  - batch system lsf
  - debian sarge, etch32 & etch64
  - including
    80 2*4core 2.67GHz Xeon with
    4*500 GB internal disk
  - ~15 used as PROOF cluster
  - = GSIAF

~ 500 TB in file server
  - 3U 15*500GB SATA, RAID 5
  - ~ 50 AliEn storage element
  - ~ 450 lustre as cluster file system

data import via AliEn SE
movement to lustre or PROOF via staging scripts
GSI / FAIR

ALICE Computing

PROOF on the Grid
GSI is a Tier-2 Centre for ALICE, one of the LHC experiments

Main Contributions from Germany:

Uni Heidelberg
Uni Frankfurt
Uni Münster
Uni Darmstadt
GSI

TPC
TRD
HLT

GridKa Tier-1
GSI Tier-2
ALICE computing model

CERN
Does: first pass reconstruction
Stores: one copy of RAW, calibration data and first-pass ESD’s

T1
Does: reconstructions and scheduled batch analysis
Stores: second collective copy of RAW, one copy of all data to be kept, disk replicas of ESD’s and AOD’s

T2
Does: simulation and end-user interactive analysis
Stores: disk replicas of AOD’s and ESD’s

Three kinds of data analysis

Fast pilot analysis of the data “just collected” to tune the first reconstruction at CERN Analysis Facility (CAF)
End-user interactive analysis using PROOF or GRID (AOD and ESD) GSIAF, glitePROOF
Scheduled batch analysis using GRID (Event Summary Data and Analysis Object Data)
Data reduction in ALICE

10^8 HI Events

- Requires AliRoot+AliEn
- Has to run on a disconnected laptop

- RAW 12.5MB/ev
- Tag 2kB/ev
- ESD 2.5MB/ev

- Reco T0/T1s

10^9 pp Events

- RAW 1 MB/ev
- Tag 2kB/ev
- ESD 40kB/ev

- Cond Data

- AODs 250kB/ev
- Analysis T0/T1s/T2/T3/laptop

- AODs 5kB/ev

~ 2 PBytes
~ 200-300 TBytes
Analysis requires only a few libraries on top of ROOT:

libSteerBase, libESD, libAOD, ...
AliEn for the File/Tag DB
GSI / FAIR

PROOF

ALICE Computing

PROOF on the Grid

[Diagram showing PROOF architecture with root nodes and grid connections]
PROOF: Parallel ROOT Facility

Interactive parallel analysis on a local cluster
    Parallel processing of (local) data (trivial parallelism)
    Fast Feedback
    Output handling with direct visualization
Not a batch system, no Grid

The usage of PROOF is transparent
    The same code can be run locally and in a PROOF system
    (certain rules have to be followed)

~ 1997 : First Prototype
    Fons Rademakers
2000...: Further developed by MIT Phobos group
    Maarten Ballintijn, ...
2005...: Alice sees PROOF as strategic tool
    2007...: Gerri Ganis, ...

~ 60 participants, most from Alice, individuals from other exp.
The PROOF Schema

Client – Local PC

- root
  - ana.C
  - Data

Remote PROOF Cluster

- root
  - node1
  - node2
  - node3
  - node4

- Result
  - Data

Proof master

Proof slave

stdout/result
The PROOF approach in a nutshell

**catalog**

**files**

**query**

PROOF job: data file list, myAna.C

**final outputs (merged)**

**feedbacks (merged)**

PROOF farm perceived as extension of local PC

same syntax as in local session

dynamic use of resources

real time feedback

automated splitting and merging
Run a task locally (from ALICE Offline Tutorial)

Start ROOT

Try the following lines and once they work add them to a macro run.C (enclose in {})

Load needed libraries

- gSystem->Load("libTree");
- gSystem->Load("libSTEERBase");
- gSystem->Load("libAOD");
- gSystem->Load("libESD");
- gSystem->Load("libANALYSIS");
Run a task locally (2)

Create the analysis manager
- `mgr = new AliAnalysisManager("mgr");`

Create the analysis task and add it to the manager
- `gROOT->LoadMacro("AliAnalysisTaskPt.cxx++g");`
  
  "+" means compile; "g" means debug
- `task = new AliAnalysisTaskPt;`
- `mgr->AddTask(task);`

Add the ESD handler (to access the ESD)
- `AliESDInputHandler* esdH = new AliESDInputHandler;`
- `mgr->SetInputEventHandler(esdH);`
Run a task locally (3)

Create a chain
- gROOT->LoadMacro("CreateESDChain.C");
- chain = CreateESDChain("ESD82XX_30K.txt", 20);

Attach the input (the chain)
- cInput = mgr->CreateContainer("cInput", TChain::Class(),
  AliAnalysisManager::kInputContainer);
- mgr->ConnectInput(task, 0, cInput);

Create a place for the output (a histogram: TH1)
- cOutput = mgr->CreateContainer("cOutput", TH1::Class(),
  AliAnalysisManager::kOutputContainer, "Pt.root");
- mgr->ConnectOutput(task, 0, cOutput);

Enable debug (optional)
- mgr->SetDebugLevel(2);
Run a task locally (4)

Initialize the manager
- mgr->InitAnalysis();

Print the status (optional)
- mgr->PrintStatus();

Run the analysis
- mgr->StartAnalysis("local", chain);
Running a task in PROOF

Copy run.C to runProof.C
Add connecting to the cluster
  ➢ TProof::Open("lxb6046")
Replace the loading of the libraries with uploading the packages
  ➢ gProof->UploadPackage("STEERBase")
  ➢ gProof->EnablePackage("STEERBase")
Same with AOD, ESD, ANALYSIS
Replaced the loading of the task with
  ➢ gProof->Load("AliAnalysisTaskPt.cxx++g")
Replace in StartAnalysis
  ➢ "local" with "proof"
Run it!
Increase the number of files to 200
Progress dialog

Executing on PROOF cluster "lx6046.cern.ch" with 33 parallel workers:
Selector: TMySelector.cxx
100 files, number of events 10000, starting event 0

Initialization time: 0.9 secs
Processed: 10000 events (1790.14 MBs) in 4.9 sec
Processing rate: 2034.0 evts/sec (364.1 MBs/sec)

Abort query and view results up to now
Abort query and discard results
Show log files
Show processing rate
How to create a PROOF Cluster

Add connecting to the cluster

```cpp
TProof::Open("lxb6046")
```

A PROOF Cluster is a set of demons waiting to start PROOF processes (master, or worker)

It can be setup

1. statically by the system administrator
   e.g. CERNAF, GSIAF,...
2. by the user
   on machines where he can login multiple processes on a multicore laptop at GSI we have scripts for our batch system
3. via gLitePROOF on the GRID

20 files

200 files
gLitePROOF: a gLite PROOF package

A number of utilities and configuration files to implement a PROOF distributed data analysis on the gLite Grid.

Built on top of RGLite:
TGridXXX interface are implemented in RGLite for gLite MW.
ROOT team accepted our suggestions to TGridXXX interface.

gLitePROOF package
It setups “on-the-fly” a PROOF cluster on gLite Grid.
It works with mixed type of gLite worker nodes (x86_64, i686...)
It supports reconnection.

http://www-linux.gsi.de/~manafov/D-Grid/docz/
RGLite example

```c++
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Submitting a Job to gLite Grid
TGridJob *job = gGrid->Submit("JDLs/proofd.jdl");
// querying a Status of the Job
TGridJobStatus *status = job->GetJobStatus();
status->GetStatus();
// Getting a Job's output back to the user
job->GetOutputSandbox("/home/anar/");
```

```c++
// Initializing RGLite plug-in
TGrid::Connect("glite");
// Changing current File Catalog directory to "dteam"
gGrid->Cd("dteam");
// Querying a list of files of the current FC directory
TGridResult* result = gGrid->Ls();
// Printing the list out
Int_t i=0;
while (result->GetFileIndex(i))
    cout << "File " << result->GetFileName(i++));
```

- Job submission,
- status querying,
- output retrieving.
- Changing file catalog directory,
- querying lists of files.
ROOT Version 5.19/02 Release Notes

ROOT version 5.19/02 has been released March 15, 2008. In case you are upgrading from version 5.14, please read the releases notes of version 5.16 and version 5.18 in addition to these notes.

Binaries for all supported platforms are available at:


Versions for AFS have also been updated. See the list of supported platforms:

http://root.cern.ch/Welcome.html

For more information, see:

http://root.cern.ch

RGLITE: A ROOT GRID Interface

RGLite plug-in - a ROOT plug-in module, which implements the ROOT Grid interface and offers to ROOT users possibilities to perform a number of operations using gLite middleware from within ROOT.

Supported features:

- Workload Management System operations:
  - job submission – normal, DAG and parametric jobs (gLite WMProxy API),
  - smart look-up algorithm for WMP Endpoints,
  - job status querying (gLite LB API),
  - job output retrieving (Globus GridFTP).
- File Catalog operations (gLite/LCG LFC API):
  - smart session manager,
  - set/query the current working catalog directory,
  - list files, directories and their attrs,
  - add/remove files in a catalog namespace,
  - add/remove directories,
  - add/remove replicas from a given file.
- An executive logging
- Support of an external XML configuration file with according XML schema.
**gLitePROOF components:**

**PROOFAgent** – a lightweight, standalone C++ application. Acts as a multifunctional proxy client/server and helps to use proof/xrootd on the Grid worker nodes behind a firewall.

**PAConsole** – a standalone C++ application, provides a GUI and aims to simplify the usage of PROOFAgent and gLitePROOF configuration files. PAConsole uses GAW to perform gLite job submissions. Users can control jobs directly using ROOT and RGLite plug-in instead of using PAConsole.

**xpdl.cfg** – a generic XROOTD configuration file (configures redirector and remote Grid workers)

**Server_gLitePROOF.sh** – a server side script. Helps to start/stop services of gLitePROOF. Could be used via command line or PAConsole GUI.

**gLitePROOF.jdl** – a JDL file, describes a generic, parametric Grid job, which is submitted to gLite and aims to execute gLitePROOF workers on Grid worker nodes.

**gLitePROOF.sh** – a job script. Executed by LRMS on remote workers. Script makes environment recon, uploads necessary packages and starts gLitePROOF services.
Content of gLite job:
gLitePROOF.jdl
gLitePROOF.sh
xpd.cfg (generic XROOTD config)
PROOFAgent (worker mode)
proofagent.cfg.xml

Workspace prerequisites:
gLite WN
ROOT
XROOTD
Outgoing connection

Workspace content:
gLite UI
ROOT
XROOTD (with GSI authentication)
xpd.cfg (generic XROOTD config)
PROOFAgent (master mode)
proofagent.cfg.xml
Server_gLitePROOF.sh
PAConsole (optional)
PAConsole: a GUI to setup a PROOF Cluster on demand
Workers on different sites

- Master: `ds2pc218.gsi.de`
- Workers:
  - `dgtestusr005@grid-wn0007.desy.de:49047` (redirect to `master`)
  - `dgr30016@linuxoc33.rz.RWTH-Aachen.DE:59946` (redirect to `worker`)
  - `dgdt0035@juggle03.zam.kfa-juelich.de:53427` (redirect to `worker`)
Summary & Observations

ALICE sees PROOF as strategic tool for prompt data analysis on their Central Analysis Facility.

Current focus is on local farms and multi-core, multi-disk desktops.

The usage of PROOF is transparent, the same code can be run locally and in PROOF.

For CPU-bound jobs we see a nearly linear speed-up. Optimal set-up in respect to IO-bound jobs still under investigations.

At GSI we operate GSIAF - a PROOF cluster for fast interactive analysis.

We developed a package to set-up a PROOF cluster on demand on the Grid. First tests are very promising.
**Transfer requests of GSI**

**runs to be transferred to GSI**

remark: weight 5 ==> highest priority

<table>
<thead>
<tr>
<th>Run</th>
<th>Requestor</th>
<th>Content (TPC/TRD)</th>
<th>weight (1-5)</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>11825</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE*</td>
</tr>
<tr>
<td>11947</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE*</td>
</tr>
<tr>
<td>12021</td>
<td>Jochen</td>
<td>TRD</td>
<td></td>
<td>MISSING</td>
</tr>
<tr>
<td>12107</td>
<td>MinJung</td>
<td>global</td>
<td></td>
<td>MISSING</td>
</tr>
<tr>
<td>12136</td>
<td>MinJung</td>
<td>global</td>
<td></td>
<td>DONE*</td>
</tr>
<tr>
<td>12163</td>
<td>Jochen</td>
<td>global</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12170</td>
<td>Jochen</td>
<td>global</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12360</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12364</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12365</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12592</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12632</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>12638</td>
<td>MinJung</td>
<td>TRD</td>
<td></td>
<td>DONE</td>
</tr>
<tr>
<td>13144</td>
<td>M. Ivanov</td>
<td>TPC cosmos</td>
<td></td>
<td>MISSING</td>
</tr>
<tr>
<td>13149</td>
<td>M. Ivanov</td>
<td>TPC cosmos</td>
<td></td>
<td>MISSING</td>
</tr>
<tr>
<td>13151</td>
<td>M. Ivanov</td>
<td>TPC cosmos</td>
<td>5</td>
<td>DONE</td>
</tr>
<tr>
<td>13165</td>
<td>M. Ivanov</td>
<td>TPC cosmos</td>
<td>5</td>
<td>DONE</td>
</tr>
</tbody>
</table>
**Terminology**

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

**Master**
PROOF machine coordinating work between slaves

**Slave/Worker**
PROOF machine that processes data

**Query**
A job submitted from the client to the PROOF system.
A query consists of a selector and a chain

**Selector**
A class containing the analysis code
In ALICE we use the Analysis Framework, therefore a AliAnalysisTask is sufficient

**Chain**
A list of files (trees) to process
Classes derived from TSelector can run locally and in PROOF

- **Begin()** once on your client
- **SlaveBegin()** once on each Slave
- **Init(TTree* tree)** for each tree
- **Process(Long64_t entry)** for each event
- **SlaveTerminate()**
- **Terminate()**
Classes TTree / TChain

A tree is a container for data storage.
It consists of several branches.

- These can be in one or several files.
- Branches are stored contiguously (split mode).
- When reading a tree, certain branches can be switched off → speed up of analysis when not all data is needed.

Compressed

A chain is a list of trees (in several files).
Loading packages

PAR file – Proof ARchive. Like Java JAR
    GZipped tar file
    PROOF-INF directory:
        BUILD.sh, building the package, executed per Slave
        SETUP.C, set environment, load libraries, executed per Slave

API to manage and activate packages:
    UploadPackage("package.par")
    EnablePackage("package")
    ShowPackages()
    ClearPackages()
Abstract

This presentation discusses activities at GSI to support interactive data analysis for the LHC experiment ALICE.

– In the computing model of Alice three kinds of data analysis are foreseen. First fast pilot analysis of the data just collected to tune the reconstruction at the CERN Analysis Facility (CAF). Second the end-user analysis using PROOF or Grid and last scheduled batch analysis using analysis trains on the Grid.

GSI is involved in the Worldwide LHC Computing Grid (WLCG) as a Tier-2 centre for ALICE.

– One focus at GSI is a setup where it is possible to dynamically switch the resources between the jobs from the Grid and a PROOF farm for fast interactive analysis via PROOF. the GSI Analysis Facility (GSIAF).

– The second emphasis is on developing a software package RGlite, an interface between ROOT and gLite which creates the possibility to create PROOF clusters on demand via standard Grid jobs.