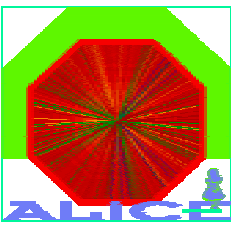


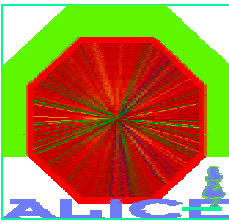
PROOF & GRID Update

Fons Rademakers



Parallel ROOT Facility

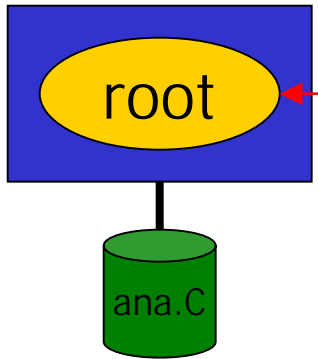
- The PROOF system allows:
 - parallel execution of scripts
 - parallel analysis of trees in a set of files
 - parallel analysis of objects in a set of fileson clusters of heterogeneous machines
- Its design goals are:
 - transparency, scalability, adaptability
- Prototype developed in 1997 as proof of concept (only for simple queries resulting in 1D histograms)



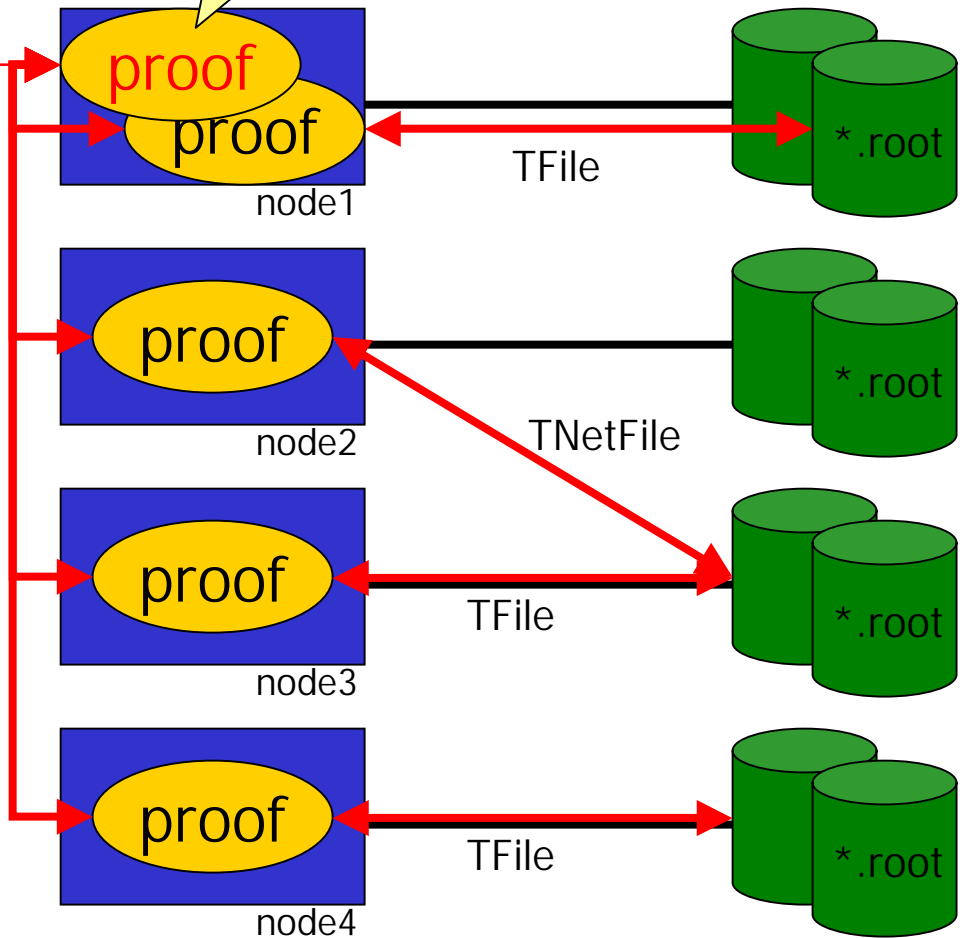
Parallel Script Execution

```
#proof.conf
slave node1
slave node2
slave node3
slave node4
```

Local PC



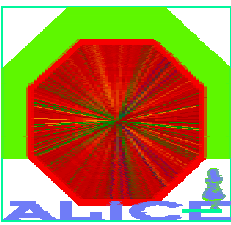
Remote Hadoop Cluster



← stdout/obj
ana.C →

```
$ root
root [0] .x ana.C
root [1] gROOT->Proof("remote")
root [2] gProof->Exec(".x ana.C")
```

proof = master server
proof = slave server

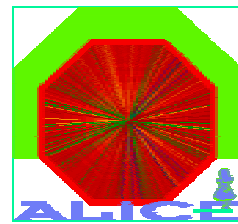


Running a PROOF Job

```
gROOT->Proof();  
TDataSet *treeset = new TDataSet("TTree", "AOD");  
treeset->Add("lfn:/alien.cern.ch/alice/prod2002/file1");  
.  
.  
.  
gProof->Process(treeset, "myselector.C");
```

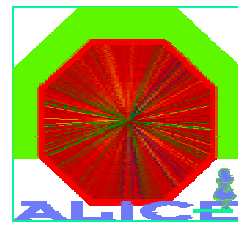
```
gROOT->Proof();  
TDataSet *objset = new TDataSet("MyEvent", "*", "/events");  
objset->Add("lfn:/alien.cern.ch/alice/prod2002/file1");  
.  
.  
.  
objset->Add(set2003);  
gProof->Process(objset, "myselector.C");
```

Different PROOF Scenarios – Static, stand-alone



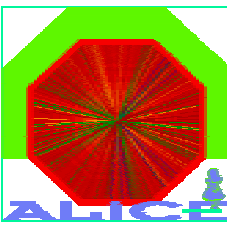
- This scheme assumes:
 - no third party grid tools
 - remote cluster containing data files of interest
 - PROOF binaries and libs installed on cluster
 - PROOF daemon startup via (x)inetd
 - per user or group authentication setup by cluster owner
 - static basic PROOF config file
- In this scheme the user knows his data sets are on the specified cluster. From his client he initiates a PROOF session on the cluster. The master server reads the config file and fires as many slaves as described in the config file. User issues queries to analyse data in parallel and enjoy near real-time response on large queries.
- Pros: easy to setup
- Cons: not flexible under changing cluster configurations, resource availability, authentication, etc.

Different PROOF Scenarios – Dynamic, PROOF in Control



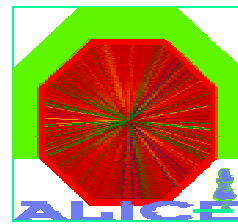
- This scheme assumes:
 - grid resource broker, file catalog, meta data catalog, possible replication manager
 - PROOF binaries and libraries installed on cluster
 - PROOF daemon startup via (x)inetd
 - grid authentication
- In this scheme the user queries a metadata catalog to obtain the set of required files (LFN's), then the system will ask the resource broker where best to run depending on the set of LFN's, then the system initiates a PROOF session on the designated cluster. On the cluster the slaves are created by querying the (local) resource broker and the LFN's are converted to PFN's. Query is performed.
- Pros: use grid tools for resource and data discovery. Grid authentication.
- Cons: require preinstalled PROOF daemons. User must be authorized to access resources.

Different PROOF Scenarios – Dynamic, AliEn in Control



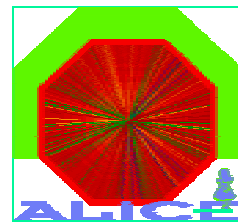
- This scheme assumes:
 - AliEn as resource broker and grid environment (taking care of authentication, possible via Globus)
 - AliEn file catalog, meta data catalog, and replication manager
- In this scheme the user queries a metadata catalog to obtain the set of required files (LFN's), then hands over the PROOF master/slave creation to AliEn via an AliEn job. AliEn will find the best resources, copy the PROOF executables and start the PROOF master, the master will then connect back to the ROOT client on a specified port (callback port was passed as argument to AliEn job). In turn the slave servers are started again via the same mechanism. Once connections have been setup the system proceeds like in example 2.
- Pros: use AliEn for resource and data discovery. No pre-installation of PROOF binaries. Can run on any AliEn supported cluster. Fully dynamic.
- Cons: no guaranteed direct response due to the absence of dedicated "interactive" queues.

Different PROOF Scenarios – Dynamic, Condor in Control



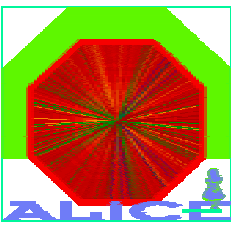
- This scheme assumes:
 - Condor as resource broker and grid environment (taking care of authentication, possible via Globus)
 - Grid file catalog, meta data catalog, and replication manager
- This scheme is basically same as previous AliEn based scheme. Except for the fact that in the Condor environment Condor manages free resources and as soon as a slave node is reclaimed by its owner, it will kill or suspend the slave job. Before any of those events Condor will send a signal to the master so that it can restart the slave somewhere else and/or re-schedule the work of that slave on the other slaves.
- Pros: use grid tools for resource and data discovery. No pre-installation of PROOF binaries. Can run on any Condor pool. No specific authentication. Fully dynamic.
- Cons: no guaranteed direct response due to the absence of dedicated "interactive" queues. Slaves can come and go.

TGrid Class – Abstract Interface to AliEn



```
class TGrid : public TObject {
public:
    virtual Int_t          AddFile(const char *lfn, const char *pfn) = 0;
    virtual Int_t          DeleteFile(const char *lfn) = 0;
    virtual TGridResult *GetPhysicalFileNames(const char *lfn) = 0;
    virtual Int_t          AddAttribute(const char *lfn,
                                       const char *attrname,
                                       const char *attrval) = 0;
    virtual Int_t          DeleteAttribute(const char *lfn,
                                       const char *attrname) = 0;
    virtual TGridResult *GetAttributes(const char *lfn) = 0;
    virtual void          Close(Option_t *option="") = 0;
    virtual const char *GetInfo() = 0;
    const char *GetGrid() const { return fGrid; }
    const char *GetHost() const { return fHost; }
    Int_t GetPort() const { return fPort; }
    virtual TGridResult *Query(const char *query) = 0;
    static TGrid *Connect(const char *grid, const char *uid = 0,
                        const char *pw = 0);
    ClassDef(TGrid,0) // ABC defining interface to GRID services
};
```

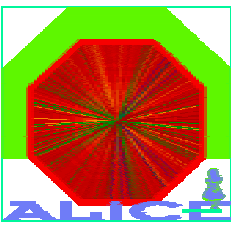
Running PROOF Using AliEn



```
TGrid *alien = TGrid::Connect("alien://alien.cern.ch", rdm);
TGridResult *res;
res = alien->Query("lfn:///alice/simulation/2001-04/V0.6*.root");
TDataSet *treeset = new TDataSet("TTree", "AOD");
treeset->Add(res);

gROOT->Proof();
gProof->Process(treeset, "myselector.C");

// plot/write objects produced in myselector.C
. . .
```



Varia...

- New developments in ROOT GUI (TG) classes by Valeriy Onuchin
 - removal on limitations of number of items in browsers
- Kerberos5 authentication support
 - By Johannes Muelmenstaedt of FNAL and MIT
- Image processing classes
 - By Reiner Rohlf of ISDC