RGLite and gLitePROOF projects

GAW Library

The both projects RGLite and gLitePROOF do not use the API of gLite directly. Instead a glite-api-wrapper (GAW) library which provides all necessary information and functionality has been created and used (Fig 1).

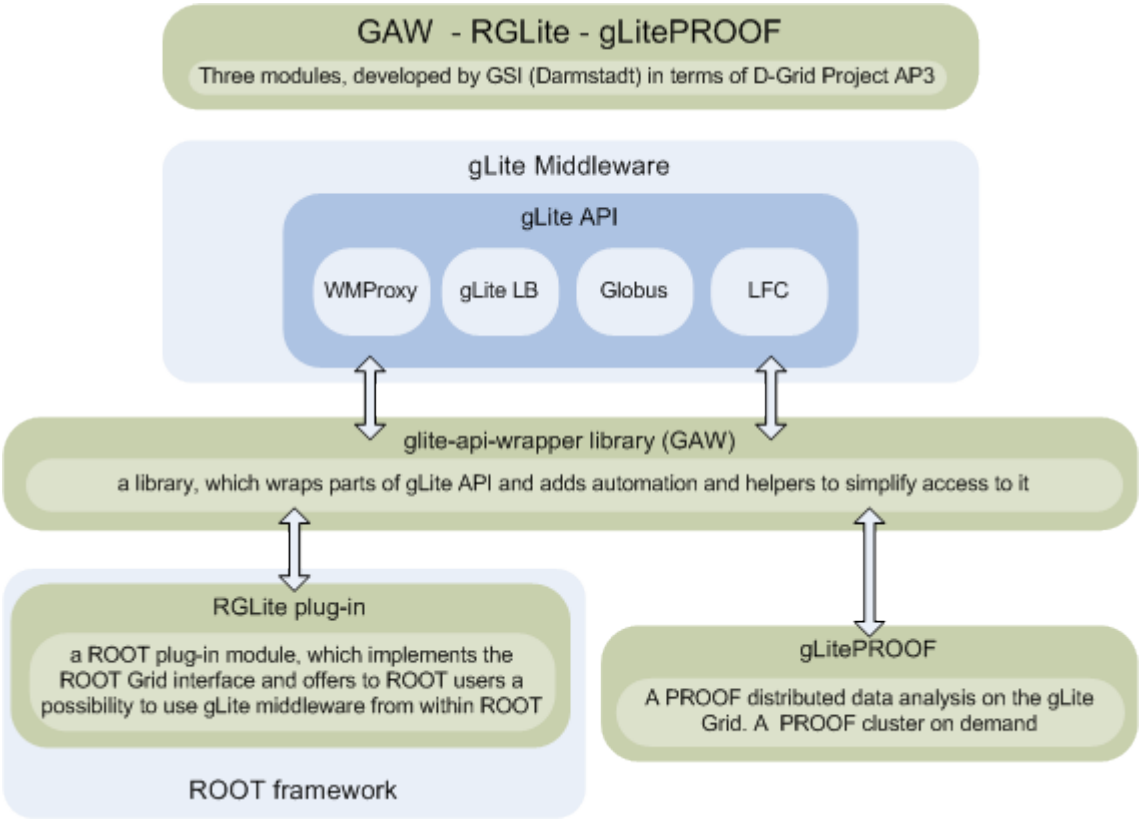

Fig 1: An overview.

The GAW library is a C++ library which wraps the parts of the gLite API which is of interest for the RGLite and gLitePROOF projects. However, GAW is not intended to be a general gLite API wrapper. Next to the wrapping functionality the GAW library additionally provides automation and helpers in order to simplify access to the required functionality of gLite. This includes detailed logging, and support of an external xml configuration file, a grid jobs manager, a catalog manager, automatic WMProxy look up methods and more.

In Fig 2 a short example usage of the GAW library is displayed. The listing shows how initialisation, job submission, as well as retrieval of job status and output can be achieved in only a few lines of code. For the same operations, in comparison, one would need several hundred lines of code when using the native gLite API directly.

GAW has a very small number of requirements and doesn't require anything which standard gLite User Interface doesn't provide. This helps to keep it easy be installed on any environment where gLite UI runs.

```
…
// GAW Init
CGLiteAPIWrapper::Instance().Init();
// Job submission
CJobManager &mng( CGLiteAPIWrapper::Instance().GetJobManager() );
mng.DelegationCredential();
const string jdl( "test.jdl" );
string jobID;
mng.JobSubmit(jdl, &jobID);
…
// Checking job's status
const glite::lb::JobStatus::Code status = mng.JobStatus(jobID);
…
// Job's output
mng.JobOutput(jobID, output_dir, &joboutput_path);
…
```

Fig 2: A short example usage of the gLite API wrapper library(GAW). In a few lines of code initialisation, job submission, status and output retrieval is accomplished.

RGLite

ROOT offers a set of abstract base classes, which provide an interface between ROOT and the Grid. These classes are TGrid, TGridJDL, TGridJob, TGridJobStatus, TGridResult, and TGridCollection. TGrid itself defines an interface to common Grid services. To open a connection to a Grid from inside ROOT, e.g., the static method TGrid::Connect() can be used. Depending on the desired Grid flavor an appropriate plug-in library is loaded to provide the real interface. TGridJDL is used to generate JDL files and for job submission to the Grid. TGridCollection manages file collections on the Grid, and TGridResult contains the result of a Grid query, for example the content of a directory in a Grid file catalogue (Fig 3). The functionality of the remaining classes should be self explanatory.

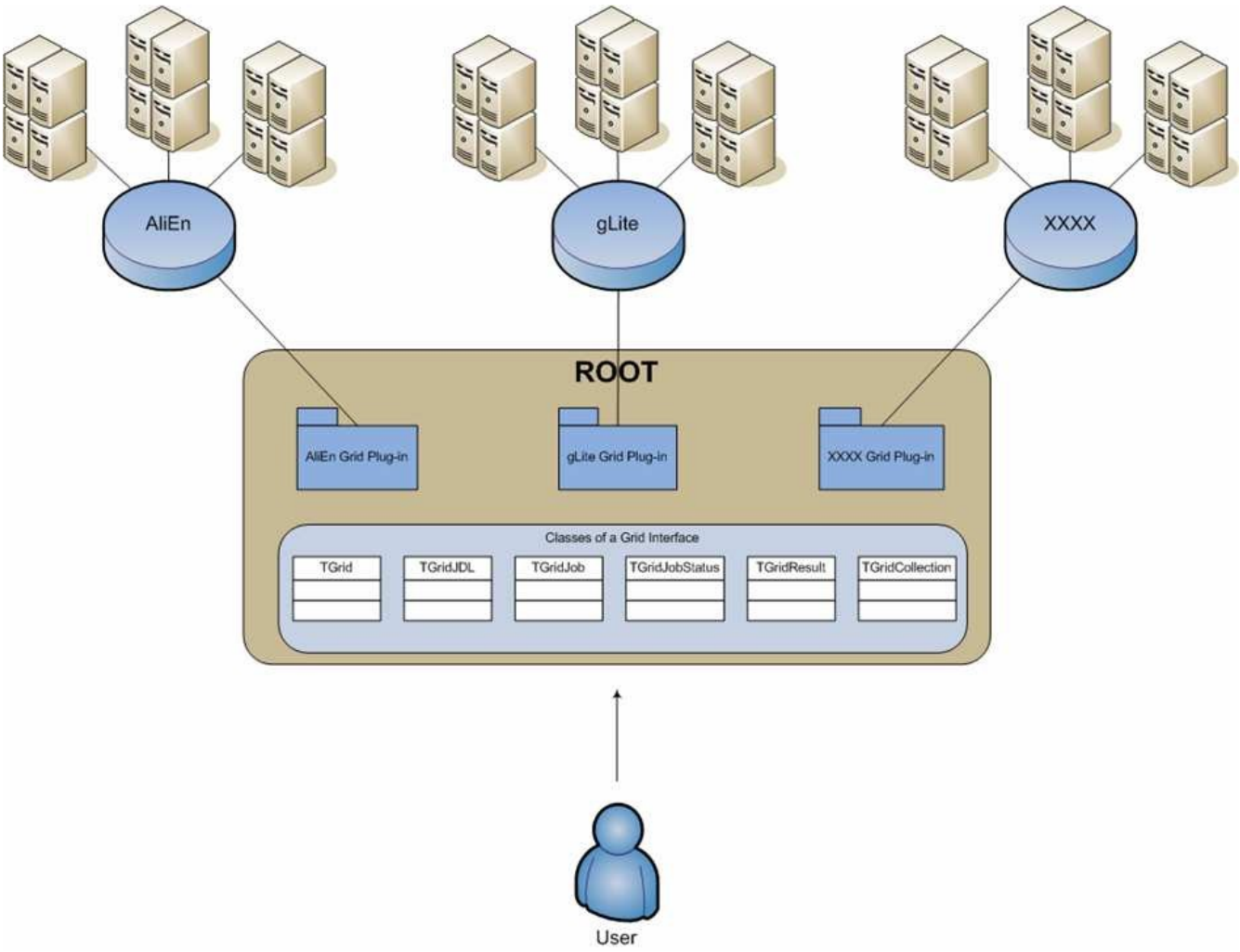

Fig 3: The abstract base classes providing the ROOT Grid interface.

A concrete implementation, TAliEnxxx, exists already for the ALICE Grid environment AliEn, and is part of the ROOT framework. An interface to the more general Grid middleware gLite has been created. The ROOT plugin, RGLite, provides the whole range of functionality, which is defined by the ROOT Grid interface. This offers ROOT users the possibility to use gLite directly from within their ROOT macros and executable. The interface includes querying the file catalogue, submitting jobs, inquiring the status, and receiving the output of jobs. Since the standard file operation mechanism in gLite is GFAL and the GFAL support is already included as part of the standard ROOT installation it has not been necessary to include this functionality in RGLite. The abstract base classes providing the ROOT Grid interface. Concrete implementations exist for the ALICE Grid environment AliEn and for gLite.

An example usage of the RGLite interface from a ROOT shell can be seen in Fig 4. In the given example a user connects to gLite, changes to the chosen directory of the used Grid file catalogue and lists the content of that directory.

```
// Initializing RGLite plug-in
TGrid::Connect( "glite" );
// Changing current File Catalog directory to "dech"
gGrid->Cd( "dech" );
// Querying a list of files of the current FC directory
TGridResult *result = gGrid->Ls();
// Printing the list out, including full file information
result->Print( "all" );
```

Fig 4: Example usage of the RGLite interface from within ROOT.

We glad to mention that RGlite is the official part of the ROOT distributive since ROOT v5.19.02.

gLitePROOF

In order to perform a PROOF analysis on the Grid, the gLitePROOF package has been developed to support interactive analysis of large data sets distributed over several sites.
gLitePROOF - is a PROOF cluster on the Grid!
gLitePROOF – is a PROOF cluster on demand!
gLitePROOF consists of a number of utilities and configuration files which have been developed at GSI in terms of the D-Grid project. The main components are PROOFAgent and PAConsole.

- PROOFAgent is a lightweight, standalone C++ application. It acts as a multifunctional proxy client and server and helps in using proof and xrootd processes behind firewalls of remote sites. Also PROOFAgent provides a number of additional functionalities which help to start, process and control an interactive PROOF analysis. The default configuration file is included in the package.
- PAConsole, also a standalone C++ application, provides a graphical user interface to simplify the usage of PROOFAgent and gLitePROOF configuration files. PAConsole uses the GAW library to provide gLite job submission functionality. A user can therefore control PROOF jobs either from within ROOT by using the RGLite plug in or via the GUI provided by PAConsole. PAConsole also provides ability to have several jobs monitored in same time. glitePROOF uses a parametric jobs submission, that means when a job is submitted there will be one parent and several (defined) child jobs, PAConsole therefore gives users an ability to submit several parametric jobs and monitor all of them. This is useful if users wants dynamically add/remove workers.

Further to the main components described above, the gLitePROOF package provides a server side script for stopping and starting the gLitePROOF services, a generic XRootd configuration file for configuring the XRootd redirector and the remote Grid workers, as well as a JDL file which describes a Grid job aiming to execute gLitePROOF workers on clusters connected to the Grid.
At the time of writing of this paper gLitePROOF version 2.0.6 has been released. This version is considered to be final stable and has been heavily tested against major DECH sites, such as FZK, DESY, UNI-Hannover, Wupertal and more.
There is one important requirement we keep while developing gLitePROOF it is to make it as much as possible to be user friendly. Starting from installation of gLitePROOF and up to the point of shutting down its services. We therefore provide simple installation procedure, detailed user manual, stable and smart services, and GUI to make our users actually care only about their analysis code.

Use Case

The main use case of the gLitePROOF package is to set up a distributed PROOF cluster on the gLite Grid. The cluster can be distributed over several sites. The only precursor is that all sites have to be interconnected via the gLite Grid middleware. A schematic picture of this use case can be seen in Fig 5.

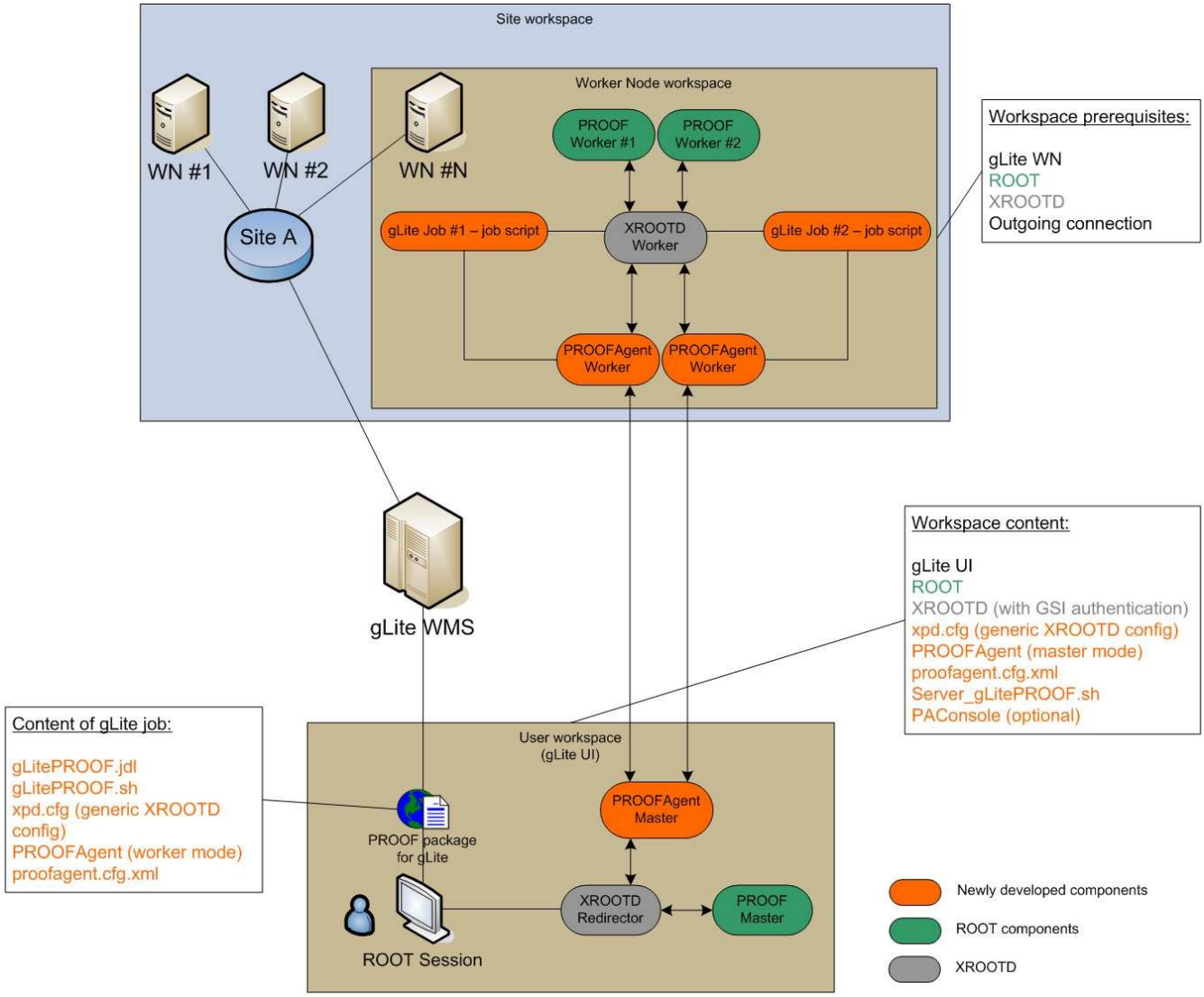

Fig 5: Setting up a distributed PROOF cluster by using thegLitePROOF package. The PROOFAgent clients are submitted as gLite Grid jobs, connect to the PROOFAgent master, and xrootd and proofd processes are started according to an automatically generated configuration file. The user can then start the PROOF analysis.

The first thing a user has to do is to start the server side processes on a central machine (a User Interface). These are the PROOFAgent server and the XRootd redirector. The PROOFAgent server then connects to the PROOF port of the XRootd redirector and waits for PROOFAgent clients to connect.

These processes can conveniently be started by using PAConsole (Simply to say, all users needs to use

is PAConsle, vie which one can manage gLitePROOF sessions completely).

Then the user submits the predefined gLitePROOF parametric job to the Grid via the PAConsole. By specifying the desired parameters in the JDL file the jobs should arrive at a site where the data to be analysed are stored. As soon as a job arrives at a remote worker node it automatically configures the environment and starts all needed client services including an XRootd data server and a PROOFAgent client. The PROOFAgent client on the remote cluster, possibly behind a firewall, communicates with the PROOFAgent server, exchanges environment data, and initiates a network tunnel. After having accepted a client connection, the PROOFAgent server adds the new node to the PROOF configuration file and the client is ready to be used as a PROOF worker. In case the PAConsole is used as session management tool, each new connection is immediately reflected in the GUI. When the instantiated PROOF workers of all submitted gLitePROOF jobs are connected or when the user is satisfied with the number of connected worker processes, the PROOF analysis can be processed as if on a local batch farm. The user then starts a ROOT session, e.g. on the private laptop, connects to the PROOF master, registers the data, and runs the analysis script. In this scenario the PROOF master can run also as one of the Grid jobs on a remote cluster. Since PROOFAgent can manage disconnects, the user can also disconnect from ROOT, restart the ROOT session and reconnect to the same PROOF cluster without having to resubmit the gLitePROOF jobs.