Knowarc_D3.2-1_08
4/8/2008

The ARC gLite Gateway

Ivan Marton[1]
Peter Stefan[2]

[1]martoni@niif.hu
[2]stefan@niif.hu

Contents

1 Introduction

The purpose of this document is to describe how Advanced Resource Connector version 1 (ARC1[3]) users can execute grid jobs on EGEE CREAM2[4] resources.

In sections 1 and 2 a brief introduction on interoperability issues will be given. Section 4 describes how to build the gateway library, section 5 shows how it can be used. Section 6 gives a tiny developer-oriented introduction on how the gateway library is built up, while section 7 addresses the current issues with the code. Section 8 concludes the technical document.

2 Interoperability

Interoperability has been one of the most important disciplines in recent grid middleware development. This term is often used in the sense of accessing resources operated by one kind of grid middleware (e.g. gLite) from a user interface operated by another kind (e.g. ARC1). There are two main possibilities to achieve this result:

- either to use a special infrastructure element, the gateway, which performs a full data structure and protocol translation between the two inter-operating grid middleware solutions,
- or to use standard or close-to-standard interfaces on both sides.

The latter solution which is preferred from standardization point of view and also has been chosen by ARC1 developers is illustrated in figure 1.

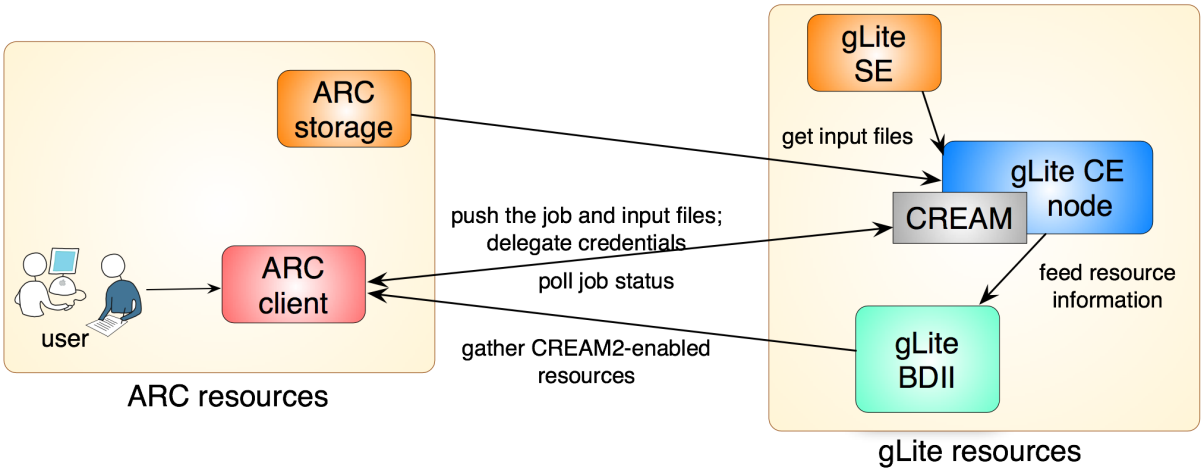

Figure 1: *This chart illustrates how the ARC1→gLite gateway library works in general. After successful authentication and credential delegation to the CREAM2 resource, users submit grid jobs through it. A wrapper script at the computing element may pull input files from different storage elements using storage protocol, say, GridFTP. Users get job information through CREAM2 and may send management commands, like kill a job or undelegate credentials.*

3 CREAM2

The main purpose of this project was to provide access to gLite CREAM2-enabled resources from ARC1 user interface, but not the other way around: i.e. accessing ARC resources from gLite is addressed in the EGEE project[5].

CREAM2 was chosen for the following reasons:

[3] http://www.nordugrid.org/
[4] http://grid.pd.infn.it/cream/
[5] http://www.eu-egee.org/

- It is a web service interface that fits to the built-in web service implementation of ARC1.
- It contains numerous improvements compared to the original CREAM interface. (Unfortunately CREAM2 is not backwards compatible with CREAM.)
- It supports job status queries compared to the other web service interfaces of gLite, like WMProxy.
- It is an official interface to gLite computing elements and as EGEE evolves it is expected that more and more computing elements will have CREAM2 interfaces.

4 Building, dependencies

The ARC→gLite gateway software is incorporated into ARC1 releases, so you only need to download either the freshest ARC1 from the source tree[6] and compile it or to install it from package[7].

The gateway library code has two dependencies:

- Globus Toolkit is needed to enable GridFTP functionality to be able to upload input files from the user interface to the CREAM computing node.
- VOMS is needed to allow users authentication on a CREAM resource.

In both cases it is highly recommended to download these pieces of software along with their NorduGrid-specific Globus[8] and VOMS[9] patches from the NorduGrid repository. In case you are interested in the details of patching, then consult the NorduGrid wiki page[10]. It might also be interesting to read how to compile Globus Toolkit. Some more details on this topic can be found on the NorduGrid Globus compilation page[11]. There are also some details on how to compile VOMS from source at NorduGrid VOMS compilation page[12].

5 Using the ARC→gLite Gateway from the Command Line

There are seven different command line tools for using the gLite gateway functions. In this section these tools will be presented in detail through examples. The job descriptions and services shown are just simple examples, they might be different in real-life usage.

Every tool has a short manual page being accessible by using the **man** on-line manual reader command or some more help can be achieved by using the <command> -? command line option.

Before getting into the details of the gatewy toolkit, the authentication and authorization processes need clarification.

In the different grid systems the most widespread authentication method is the certificate based authentication. GLite also requires VOMS-enabled proxy X509 certificates to access resources. It means that you should be a member of the Virtual Organisation that is valid on the remote site. If the VOMS package has been correctly installed then the following command provides the necessary VOMS proxy certificate.

```
$ voms-proxy-init -voms <Your Virtual Organisation>
```

For example:

```
$ voms-proxy-init -voms knowarc.eu
```

CREAM2 service URLs can directly be downloaded from any BDII database of gLite using the following LDAP command:

[6] http://svn.nordugrid.org/
[7] http://download.nordugrid.org/software/nordugrid-arc1/
[8] http://download.nordugrid.org/software/globus/
[9] http://download.nordugrid.org/software/voms/
[10] http://wiki.nordugrid.org/images/4/46/NorduGrid-globus.patch.txt
[11] http://wiki.nordugrid.org/index.php/Globus_Libraries
[12] http://wiki.nordugrid.org/index.php/VOMS_compiling_detailes

```
$ ldapsearch -x -h lcg-bdii.cern.ch -p 2170 -b o=grid '(GlueCEUniqueID=*blah*)'
```

or

```
$ ldapsearch -x -h lxbra2305.cern.ch -p 2170 -b mds-vo-name=local,o=grid //
        '(GlueCEUniqueID=*cream*)' | grep GlueCEInfoContactString | //
        awk '{print $2}' | sort | uniq
```

5.1 glitedelegate

If you want to submit a job to a gLite server, first you have to own a valid delegation on the remote site. You can either use an old, but still valid delegation identifier, or register a new one. There is a command line tool to register a delegation on the resource to be used.

The usage is very easy and straighforward. All you need is a working CREAM2 service URL, and a locally unique arbitrary delegation ID. This delegation ID will be associated to the remote resource. If you are not sure wether your favourite delegation ID is occupied or not, then try to delete it before registering it again. (See Section 5.6 for further details!) The delegation command has the following syntax:

```
$ glitedelegate <delegation ID> <service URL>
```

For example:

```
$ glitedelegate test_delegation //
        https://cream.grid.upjs.sk:8443/ce-cream/services/gridsite-delegation
```

5.2 glitesub

In the previous section it was shown how to create delegation. (See Section 5.1!) This function, the job submission, is the only one which needs the delegation ID.

The next step is to prepare the necessary input files and the job description!

The glitesub command syntax is the following:

```
$ glitesub -D <delegation ID> <service URL> <job description> <info file>
```

Here the delegation ID is the previously registered identification. The service URL will be different than in the delegation example above, because this is not the URL of the delegation, but the execution service. The job description must be in JDL format like in the example below. The info file stores information about job submitted. It should either be a non-existing file or it is overwritten. This file contains the job ID, the input sandbox URL, the output sandbox URL and the service URL. The submission command will take care of uploading the necessary input files up to the execution node.

It is important that the *VirtualOrganisation* and *QueueName* attributes should be the same as your virtual organisation name and the corresponding queue name on the server side.

An example JDL description:

```
[
Executable = "/bin/hostname";
StdInput = "std.in";
StdOutput = "std.out";
StdError = "std.err";
BatchSystem = "pbs";
VirtualOrganisation = "knowarc.eu";
InputSandbox = {"std.in"};
OutputSandbox = {"std.out","std.err"};
QueueName = "knowarc.eu";
OutputSandboxDestURI = { "gsiftp://localhost/std.out", "gsiftp://localhost/std.err" };
]
```

An example how to use this command:

```
$ glitesub -D test_delegation //
        https://cream.grid.upjs.sk:8443/ce-cream/services/CREAM2 description.jdl job.info
```

5.3 glitestat

The **glitestat** tool extracts job information about a CREAM2 job identified by the information file. This file made by command **glitesub** is the only argument needed here.

Usage:

```
glitestat <info file>
```

For example:

```
glitestat job.info
```

The possible responses and their meanings are the following:

- ACCEPTING - The job submission is comleted but the job is not yet scheduled
- SUBMITTING - Scheduling in progress
- INLRMS:Q - The job is already at the local resource manager system and it is queued
- INLRMS:R - The job is waiting at the local resource manager system for running
- INLRMS:S - The job is actually running
- KILLED - The job was terminated
- FINISHED - The job has finished
- FAILED - The job had some failure
- FAILES - The job had some failure at LRMS level
- EXECUTED - The job has been finished and there is no stating information available

5.4 glitekill

The **glitekill** command can be used for killing a remote job. The only necessary argument is the information file made by **glitesub**. (See Section 5.2 for further details!) This command initiates stopping the job. You can check the effect by using **glitestat**. (See Section 5.3 for its usage!)

Usage:

```
glitekill <info file>
```

For example:

```
glitekill job.info
```

5.5 gliteclean

Either after killing a job or because of some remote error occurs garbage files may remain on the server. These files can be purged by using command **gliteclean**. If you intentionally kill a job on the computing node, the job related stuff might also remain as garbage in the remote queue. This command also removes the job from the gLite server queue. It is important to note that this command removes the local information file as well. It is very useful to run this command after killing a job. (See Section 5.4!)

Usage:

```
gliteclean <info file>
```

For example:

```
gliteclean job.info
```

5.6 gliteundelegate

After your work is complete it is possible to unregister and delete the delegation entry from the remote site by using command **gliteundelegate**. You can also use it, if you want to register a delegation ID but are unsure whether it is used or not. Put the delegation service URL into the argument!

Usage:

```
$ gliteundelegate <delegation ID> <service URL>
```

For example:

```
$ gliteundelegate test_delegation //
        https://cream.grid.upjs.sk:8443/ce-cream/services/gridsite-delegation
```

6 Developers notes

This section describes how to use the library on your own client. The base class of the gLite client is written as an ARC Client Component (ACC) but also can be used as a standalone library. There is nothing more to be done just to link the **CREAMClient**, **arcclient** and **arccommon** libraries installed with the ARC1 sofware and to create an instance of the *CREAMClient* class.

This client object has a very easy-to-use and intuitive interface. It uses the VOMS proxy certificate to build the secure channel to the server and to sign that of the server's. Actually the client looks for this certificate just at its default location. If you want to store it somewhere else, you have to modify the source code itself.

The different functions throw *CREAMClientError* exceptions that should be caught in the application.

The following sections present the set of functions provided by the library.

6.1 CREAMClient(Arc::URL, Arc::MCCConfig)

The constructor of the class performs the necessary initialization work. It receives two arguments: the service URL and the ARC message chain component configuration file to establish the communication channel to the server. These pieces of information are indispensable to create your client and to communicate with the remote site.

6.2 setDelegationId(std::string)

Besides the constructor this is the other generic-purpose function that can be used in CREAM2 client applications. This function sets the previously registered and referred delegation ID on the client. The function is very simple: just set a private variable and returns with no value.

6.3 createDelegation(std::string)

The **createDelegation** function is about to perform the whole delegation registration process. It sends a *getProxyReq* message having the requested delegation ID, signs the received certificate and sends it back in a *putProxy* SOAP message to the server. These three steps constitute the delegation registration process. As almost every funtion, this one has no return value either. If there are any problems during the communication, either locally in the channel or at the remote site, the function throws *CREAMClientError* described previously. If there are no exceptions thrown, then the command is considered to have been successfully completed.

6.4 destroyDelegation(std::string)

This method has the reverse functionality of the **createDelegation** function. It sends the CREAM2 *destroy* message to the remote server.

6.5 submit(std::string)

The **submit** function is the most complex part of the class. It translates the job description received as the argument, registers the job with a *JobRegisterRequest* message, uploads the locally stored files, if necessary, then enables the job execution on the server by sending a *JobStartRequest* message. Finally it returns with a **creamJobInfo** object that contains a *jobId* (job indetifier), a *creamURL* (as the service URL), an *ISB* (reference to the Input Sandbox) and an *OSB* (Output Sandbox) member to describe the registered job. In the **glitesub** command (See Section 5.2 for details) these pieces of information are stored in the information file.

6.6 stat(std::string)

This method queries the job status from the gLite CREAM2 server. The return value is the job status translated to the ARC terminology. The possible values are presented in the section of the **glitestat** command line tool. (See Section 5.3!)

6.7 cancel(std::string)

The **cancel** function sends a *JobCancelRequest* SOAP message to the server and handles the emerging exceptions. It can be used for canceling a remotely registered and possibly running job.

6.8 purge(std::string)

The **purge** function sends a *JobPurgeRequest* message to server and throws an exception in case of any emerging problems.

6.9 An example gLite client

Finally, here is an example code to present how easy and simple is to write a new client. This code sample requests the job status of a previously submitted remote job, and writes it to the standard output. The client is written in C++.

```
#include "CREAMClient.h"
#include <iostream>

int main(int argc, char* argv[]){
  Arc::URL url( SERVICE_URL );
  Arc::MCCConfig cfg;

    Arc::Cream::CREAMClient gLiteClient(url,cfg);
    try {
        std::cout << "Job status: " << gLiteClient.stat( JOBID ) << std::endl;
    } catch (CREAMClienException& cce) {
        std::cerr << "ERROR: " << cce.what() << std::endl;
        return 1;
    }
    return 0;
}
```

It is of course mandatory to define *SERVICE_URL* and *JOBID* strings to their real values to reach the proper functionality.

7 Outstanding issues

Even though the gateway library code is ready for production use, there are a couple of issues to be handled in the upcoming development.

7.1 Full ARC1 client integration

Full ARC1 client integration basically means that the previously presented commands will disappear and their functionality will be available in the standard ARC1 client toolkit. However there are two dependencies to be met to achieve this stage:

- On one hand ARC1 broker along with the BDII-capable target object should be finalized.
- On the other hand a conflicting Globus Toolkit OpenSSL and the standard OpenSSL issue is to be investigated.

The latter issue was revealed when the VOMS proxy certificate delegation method was implemented in the gateway library. Since in the ARC1 client-side library there is a built-in Message Chaining Component (MCC) which already uses Globus Toolkit OpenSSL functions to enable establishing secure communication. Linking glite* tools directly against the standard OpenSSL libs caused congesting SSL functions which led to the breakdown of the whole library. The solution had been not linking the standard OpenSSL libs, just the patched OpenSSL in the Globus Toolkit.

7.2 xRSL→JDL translation and related problems

It is possible to express jobs in XRSL which have no equivalent in JDL. Specically, in XRSL it is possible for input files downloaded from storage servers to have their names changed. This is not possible in JDL. So a user might in XRSL ask for this:

```
(inputfiles=(gsiftp://interop.dcgc.dk/storage/datafile1.txtinput.txt))
```

That is not possible to express in JDL. It only allows the file to be downloaded as datafile1.txt. This could be solved by a wrapper script that renames files according to the XRSL specication. This does not solve our problem completely however. For example if a user wants to concatenate the stdout from two jobs, he would write something like the following in XRSL:

```
(inputfiles=(gsiftp://interop.dcgc.dk/storage/job1/output.txtinput1.txt)
            (gsiftp://interop.dcgc.dk/storage/job2/output.txtinput2.txt))
```

This cannot be solved just using the renaming method, because both input files would be retrieved before renaming is done and they would therefore overwrite each other.

The solution is to recognize jobs of this type and simply barring them from using gLite resources or to let such jobs fail with an error message explaining that the user has to express their job in a different way.

7.3 RunTime Environments related issues

Many grid jobs rely on software being available at the execution location. The requirement as well as the initialization of such software is expressed through the use of Run-Time Environments (RTEs). While the library is capable of translating the RTE request itself it is not capable of translating the name of the RTE. No standardized cross Grid nomenclature has been adopted, nor are the authors aware of any such activity. A temporary solution would be for each VO to have a remotely accessible list of RTEs and their names in each job description language. This could then be used for translating jobs.

8 Conclusions

The document has described an ARC1 library extension that allows ARC1 grid users to access gLite CREAM2 computing resources. The current version of the code allows direct access to resources but as the new ARC1 brokering solution comes out the gateway functionality will be integrated fully into the ARC1 client code. The same version will also be able to access the gLite information system to extract CREAM2 resource information.

A built-in xRSL→JDL translator will also enable to use purely xRSL job description on ARC1 user interfaces.