EMI user tutorial

Access the EMI UI

you have to login via ssh on a remote machine in Catania; XX is a number between 01 and 40, you will be told which actually use. Password will be also told verbally. ssh -p 2222 -l lyonXX emi-tutor.ct.infn.it `ssh -p 2222 -l lyonXX emi-tutor.ct.infn.it`

gLite

Authenticate yourself

Create your voms proxy, certificate passphrase is **LYON**

```
voms-proxy-init --voms testers.eu-emi.eu
```

and check it's valid

```
$ voms-proxy-info -all
subject   : /C=IT/O=GILDA/OU=Personal Certificate/L=LYON/CN=LYONXX/CN=proxy
issuer    : /C=IT/O=GILDA/OU=Personal Certificate/L=LYON/CN=LYONXX
identity  : /C=IT/O=GILDA/OU=Personal Certificate/L=LYON/CN=LYONXX
type      : proxy
strength  : 1024 bits
path      : /tmp/x509up_u542
timeleft  : 11:58:40
key usage : Digital Signature, Key Encipherment, Data Encipherment
=== VO testers.eu-emi.eu extension information ===
VO        : testers.eu-emi.eu
subject   : /C=IT/O=GILDA/OU=Personal Certificate/L=LYON/CN=LYON40
issuer    : /C=IT/O=INFN/OU=Host/L=CNAF/CN=emitestbed07.cnaf.infn.it
attribute : /testers.eu-emi.eu/Role=NULL/Capability=NULL
timeleft  : 11:58:40
uri       : emitestbed07.cnaf.infn.it:15002
```

Browse available resources

Through the command `lcg-infosites` we can gather the available resources for our VO (in this case `testers.eu-emi.eu`) We see first which Computing Elements are available

```
$ lcg-infosites --vo testers.eu-emi.eu ce
#   CPU      Free   Total Jobs   Running   Waiting   ComputingElement
----------------------------------------------------------------
     16        16            2         2         0   cert-09.cnaf.infn.it:8443/cream-lsf-demo
    200       159           42        42         0   cream-37.pd.infn.it:8443/cream-lsf-cert
      2       159            0         0         0   cream-37.pd.infn.it:8443/cream-lsf-creamtest
    216       159            0         0         0   cream-37.pd.infn.it:8443/cream-lsf-creamtest
      8         8            0         0         0   lxbra2308.cern.ch:8443/cream-pbs-testersemi
```

Tip : If you get sick of typing such a long VO name, shortcut it through an environment variable

Now we query the information system to know which Storage Elements are available

```
$ export MYVO="testers.eu-emi.eu"
$ lcg-infosites --vo $MYVO se
 Avail Space(kB)  Used Space(kB)  Type    SE
----------------------------------------
         7236051         1212806  SRM     cork.desy.de
              -1              -1  SRM     emitestbed09.cnaf.infn.it
        83799892        23521860  SRM     lxbra1910.cern.ch
```

```
        83944410        23377343  SRM     lxbra2502.cern.ch
        15382114          629248  SRM     lxbra2506v1.cern.ch
        59077997             942  SRM     sligo.desy.de
         7636881             n.a  SRM     xen-ep-emi-tb-se-3.desy.de
         7226745              12  SRM     xen-ep-emi-tb-se-4.desy.de
```

Store a file on a Storage Element

Create a local file, and then store it on an available SE:

```
$ echo "This a sample file" > example.txt
$ cat example.txt
This a sample file
$ lcg-cr -d emi-demo03.cnaf.infn.it file:$PWD/example.txt
 GSIFTP: default set up URL mode
GSIFTP: dest: set up FTP mode. DCAU disabled. Streams =  1, Tcp BS = 0
guid:e2edabff-3fa7-4853-b44a-9cab256befdb
```

The file has been stored on the SE `lxbra1910.cern.ch` and the lcg-cr command returns a Grid Unique Identifier (**guid**) for our file. Our file has also automatically registered in the File Catalog and assigned a Logical File Name (**lfn**). With the option `-l` we could specify an lfn for the file. The File Catalog provides an easier way to identify and browse our files using these Logical File Names.

To see our file we can use the File Catalog command lfc-ls command to list all files, in this case we will limit it to files created today.

```
$ lfc-ls /grid/$MYVO/generated/2011-09-22
file-99018d3a-138c-4344-82a0-48a2ad10c27b
```

Note that the identifier returned here is the lfn not the guid. However because you can see all files in this directory, this is not particularly useful as you have no way to identify your file. To avoid this problem you should set the lfn to some sensible value when creating the file with the `-l` option. Try creating another file with a logical file name containing your user id, for example
`lfn:/grid/$MYVO/generated/2011-09-22/lyonXX.txt`.

Now when you use the `lfc-ls` command you should be able to identify your file via its unique filename.

To copy the file from the SE to the UI we can use the `lcg-cp` command as follows

```
$ lcg-cp guid:e2edabff-3fa7-4853-b44a-9cab256befdb file.txt
```

or we could use the lfn

```
$ lcg-cp lfn:/grid/$MYVO/generated/2011-09-22/file-99018d3a-138c-4344-82a0-48a2ad10c27b file.txt
```

We can now delete the registered file using the GUID; if we check for file existence after deletion, we obviously don't find it.

```
$ lcg-del -a guid:e2edabff-3fa7-4853-b44a-9cab256befdb
$ lfc-ls /grid/$MYVO/generated/2011-09-22
```

Submit a job

Job submission request are expressed via JDL (Job Description Language). Find below a very simple but usable example, which just runs "uname -a" on the executing node

```
$ cat uname.jdl
Type = "Job";
```

```
JobType = "normal";
Executable = "/bin/uname";
StdOutput = "uname.out";
StdError = "uname.err";
OutputSandbox = {"uname.out","uname.err"};
Arguments = "-a";
requirements = other.GlueCEStateStatus == "Production";
rank = -other.GlueCEStateEstimatedResponseTime;
RetryCount = 0;
```

We now submit the job to the Workload Management System (WMS) which will find a suitable resource on which our job can run:

```
$ glite-wms-job-submit -a uname.jdl

Connecting to the service https://lxbra2303.cern.ch:7443/glite_wms_wmproxy_server


====================== glite-wms-job-submit Success ======================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://lxbra2303.cern.ch:9000/F0KY_m0DBH5wpXzLt59q5A

==========================================================================
```

On success, the submission command returns a job identifier, that we eventually use to monitor job status and, once it's done, we use the job identifier to retrieve the output

```
$ glite-wms-job-status https://lxbra2303.cern.ch:9000/F0KY_m0DBH5wpXzLt59q5A


======================== glite-wms-job-status Success ========================
BOOKKEEPING INFORMATION:

Status info for the Job : https://lxbra2303.cern.ch:9000/F0KY_m0DBH5wpXzLt59q5A
Current Status:     Done (Success)
Logged Reason(s):
    - job completed
    - Job Terminated Successfully
Exit code:          0
Status Reason:      Job Terminated Successfully
Destination:        lxbra2308.cern.ch:8443/cream-pbs-testersemi
Submitted:          Sat Jul  9 13:32:09 2011 CEST
==========================================================================


$ glite-wms-job-output https://lxbra2303.cern.ch:9000/F0KY_m0DBH5wpXzLt59q5A

Connecting to the service https://lxbra2303.cern.ch:7443/glite_wms_wmproxy_server


================================================================================

         JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
https://lxbra2303.cern.ch:9000/F0KY_m0DBH5wpXzLt59q5A
have been successfully retrieved and stored in the directory:
/tmp/jobOutput/lyon40_F0KY_m0DBH5wpXzLt59q5A

================================================================================
```

```
$ ls /tmp/jobOutput/lyon40_F0KY_m0DBH5wpXzLt59q5A/
uname.err  uname.out
$ ls -l /tmp/jobOutput/lyon40_F0KY_m0DBH5wpXzLt59q5A/
total 4
-rw-r--r-- 1 lyon40 users   0 Jul  9 13:35 uname.err
-rw-r--r-- 1 lyon40 users 116 Jul  9 13:35 uname.out
$
$ cat /tmp/jobOutput/lyon40_F0KY_m0DBH5wpXzLt59q5A/uname.out
Linux lxbra2506v6.cern.ch 2.6.18-238.12.1.el5xen #1 SMP Tue May 31 13:35:45 EDT 2011 x86_64 x86_6
```

Submitting Multiple Jobs

There are several ways to submit multiple jobs to a gLite Grid. The simplest is to submit separate jobs as a collection of jobs. To do this copy a number of jdl files into a directory, say `jdls`, for example, then submit a collection type job which points at this directory. All of the jdl files in the directory will be executed.

```
$ glite-wms-job-submit -a --collection jdls/

Connecting to the service https://lxbra2303.cern.ch:7443/glite_wms_wmproxy_server


====================== glite-wms-job-submit Success ======================

The job has been successfully submitted to the WMProxy
Your job identifier is:

https://lxbra2303.cern.ch:9000/y7eIsv-bHDpuNjE8v2Y-yw

==========================================================================
```

You can then view the status and get the output of all jobs via the single job id which is returned by `glite-wms-job-submit`.

ARC

Creating and managing proxy certificates

The `arcproxy` command is capable of creating different kind of certificates, also with VOMS extension, and it has support for MyProxy.

Note: If you have already created a proxy certificate in the gLite part of this tutorial you should be able to use the same certificate! Otherwise, follow the instructions here.

The default usage is simply to create a proxy from the user's cert and key file located at the default `~/.globus` path, or specified in the `~/.arc/client.cont`:

```
$ arcproxy
```

If you would like to have VOMS extensions:

```
$ arcproxy --voms VONAME
```

You can check your proxy with:

```
$ arcproxy -I
```

Getting information about clusters

The `arcinfo -c CLUSTERNAME` or `arcinfo -g INDEXSERVER` command prints the information about the specified cluster or the clusters registered into the index server. Without argument the command will get the URLs from the `defaultservices` specified in the `~/.arc/client.conf` file. To use the EMI testbed, we will specify the index server of the testbed in the `~/.arc/client.conf`:

```
[common]
defaultservices=index:ARC0:ldap://arc-emi.grid.upjs.sk:2135/O=Grid/Mds-Vo-Name=ARC-EMI
```

Then we can see information about the clusters registered to this index server with the `arcinfo` command:

```
$ arcinfo
```

With the `--long` argument it will print more information.

You can also specify aliases for specific clusters or index servers in the `[alias]` section of the `~/.arc/client.conf` command:

```
[alias]
test7=computing:ARC0:testbed7.grid.upjs.sk
```

Then you can query just this cluster with the `-c` option:

```
$ arcinfo --long -c test7
```

Of course you can just specify the hostname of the cluster without having an alias:

```
$ arcinfo -l -c testbed8.grid.upjs.sk
```

Here is an example to query the information system with the `-g` option:

```
$ arcinfo -g ldap://arc-emi.grid.upjs.sk:2135/O=Grid/Mds-Vo-Name=ARC-EMI
```

But because we set this URL as a default service, we don't have to specify any `-g` or `-c` arguments during this tutorial for the other commands.

Submitting a simple job

The `arcsub` commands does everything from communicating with the information systems and doing brokering to translate job descriptions and copy local input files to the cluster.

Here is a simple XRSL job description

```
$ cat <<EOF > hostname.xrsl
&(executable="/bin/hostname")
(stdout="hostname.out.txt")
EOF
```

With the `--dump` option, the command does not do the actual submission, just chooses a computing element and prints the job description to be sent to it:

```
$ arcsub hostname.xrsl --dump
```

We can really submit the job without the `--dump` option:

```
$ arcsub hostname.xrsl
```

The command prints the URL of the job, which serves also as the unique ID. This URL can be used to query the status of the job, and also to list the contents of its session directory on the cluster.

The `arcsub` command supports the JDL and the JDSL job description languages too, you can try to submit jobs using them as an exercise.

You can use the `-e` command to provide the job description as a command line argument:

```
$ arcsub -e '&(executable="/bin/hostname")(stdout="hostname.out.txt")'
```

Querying the status

The `arcstat` command queries the status of the active jobs. Without argument it doesn't know which jobs to query, so we have to specify either some job IDs, some clusters or index servers (with the `-c` and `-g` options), or the `-a` option which prints the status of all our jobs:

```
$ arcstat -a
```

Or we can just specify the ID (the URL) of the job:

```
$ arcstat JOBID
```

Getting the standard output/error

The `arccat` command prints the standard output or the standard error of a job. The same way as the `arcstat` command, we have to specify some jobs (either by job ID, or by cluster/index server), or we can give the `-a` options for all jobs. By default it prints the standard output, but we can get the standard error with the `-e` option:

```
$ arccat -a
$ arccat JOBID
$ arccat JOBID -e
```

The `-e` option only works if you specified a stderr filename in the job description, e.g.:

```
&(executable="/usr/bin/java")
(arguments="-version")
(stdout="javaversion.txt")
(stderr="error.txt")
```

Alternatively you can join the stdout and stderr into the stdout file with the `join` parameter:

```
&(executable="/usr/bin/java")
(arguments="-version")
(stdout="javaversion.txt")
(join=yes)
```

Listing the session dir of the job

The session dir is the directory where the job lives on the cluster. We can list its contents during or after the job run with the `arcls` command. We have to specify the job's URL:

```
$ arcls JOBID
```

We can copy any files out of the session dir with the `arccp` command. You just have to add a slash after the JOBID (which is a URL), and write the name of the file:

```
$ arccp JOBID/FILENAME LOCALNAME
```

where `LOCALNAME` is the local path on our machine where we want to put the file.

We can also use `-` instead of `LOCALNAME` which will copy the file to the standard output:

```
$ arccp JOBID/FILENAME -
```

(The `arcls` and `arccp` command together with the `arcrm` command provides full-featured data operations, supporting multiple protocols, registration of replicas to file catalogs, even SRM requests.)

Getting the results of a job

After the job finished we can download all the results and remove the job from the grid in one step with the `arcget` command. We have to specify which jobs we want to get the same way as we did for the `arcstat` or `arccat` commands.

```
$ arcget -a
$ arcget JOBID
$ arcget -c CLUSTER
```

The output will be downloaded to the current directory into a subdir named by the job's numerical ID. Or you can specify the `-J` option to name the directory with the jobname instead:

```
$ arcget -a -J
```

Please note that by default this command removes the job from the cluster. This can be avoided by the `--keep` option.

Input files

Most of the jobs need some input files. Some of these files are located on the local machine from where the jobs are submitted, some of them are located on remote servers. Both kinds of files can be specified in the job description, the local files will be uploaded by the `arcsub` command during job submission, the remote files will be downloaded by the computing element before the job starts.

A simple example with a local input file:

```
$ cat <<EOF > data.txt
Hello EMI!
EOF

$ cat <<EOF > simple-input.xrsl
&(executable="/bin/cat")
(arguments=data.txt)
(stdout="stdout.txt")
(inputfiles=(data.txt data.txt))
EOF

$ arcsub simple-input.xrsl
```

We can check the session dir with the `arcls` command:

```
$ arcls JOBID
```

After a couple of minutes we can check the standard output with the `arccat` command:

```
$ arccat JOBID
```

Output files

The results of the job could either be automatically copied to a remote location (by specifying the target URL in the job description) or they can be kept on the cluster and downloaded by the `arcget` client tool. If an output file is not specified in the job description as either a remote upload or a file to keep: it will be removed when the job finishes.

If we want to upload a file to a remote storage, we have to specify the target URL, if we want to keep the file, we have to specify an empty target URL in the XRSL:

```
(outputfiles=(data.txt ""))
```

The file containing the standard output (or the standard error if specified) will be always kept by default.

Simple parameter sweep

The `arcsub` command doesn't have support for parameter sweep jobs, so we have to use other methods. Here is a simple way to do it. First we create a template job description:

```
$ cat <<EOF > simple-sweep.template
&(executable="/usr/bin/seq")
(arguments=-args-)
(stdout="sweep-name-.txt")
(jobname="sweep-name-")
EOF
```

I used the `-args-` and `-name-` strings, which I will replace later with `sed`. (This is not ARC-specific.)

The `arcsub` command can accept a job description as a string with the `-e` option. We will do some bash scripting to sweep through an interval with multiple jobs. The SIZE bash variable will contain the size of our window:

```
$ SIZE=10
$ for i in `seq 100 $SIZE 300`; do arcsub -e "`sed -e "s/-args-/$i $((i+SIZE))/" -e "s/-name-/.$i
```

Because we specified the `--dump` option, this commmand just printed the job descriptions but didn't actually submit the job. If it looks right, we can remove the `--dump` option and do the submission for real:

```
$ for i in `seq 100 $SIZE 300`; do arcsub -e "`sed -e "s/-args-/$i $((i+SIZE))/" -e "s/-name-/.$i
```

Then after several minutes, we can check the status and the standard outputs:

```
$ arcstat -a
$ arccat -a
```

We can also use `grep` to wait until every job finishes:

```
$ arcstat -a | grep "State:" | grep -v "FINISHED" | wc -l
```

This will show the number of unfinished jobs.

Then we can get the outputs with the `arcget -a` command, or if the standard output satisfied us, then we can just clean the jobs with the `arcclean -a` command.

UNICORE

The UCC is already configured; you can see this in ~/.ucc/preferences (Note: You don't have to specify the password in the configuation file. If you omit the line, UCC will ask you for it on every call. To avoid typing your password repeatedly, you can run ucc shell and then issue every UCC command from within the UCC shell.)

First, you have to run the connect command:

```
$ ucc connect
```

Help for each ucc command with -h:

```
$ ucc -h
```

List available sites

```
$ ucc list-sites
```

To enter an interactive mode:

```
$ ucc shell
```

List applications and storages and exit an interactive mode:

```
> list-applications
> list-storages
> exit
```

UCC date.u

Copy this file to a date.u file:

```
# simple job: run Date
{
   ApplicationName: Date,
   ApplicationVersion: 1.0,
}
```

UCC - Running job

```
$ ucc run date.u -v
```

In this case the standard out went for example to 58c55a2d-83ec-450f-b5f7-3e6f958312f7.stdout

Get the status of a specific job using ucc get-status. As an argument you can either use the job file that you got from run -a or the End Point Reference (EPR) you got from list-jobs :

```
$ ucc run -a date.u -v -b
$ ucc list-jobs
$ ucc get-status job
$ ucc get-output job
```

UCC Data Management

Often your job will need to access some data, or you may need to upload your own executable or script with your job. Similarly, your job might produce output files which you need to retrieve after your job has

completed. To do this, the job description must contain details of the input and output files.

You can upload files directly from and download to the UI machine as follows

```
{
    Imports: [
       { From: "/path/fileName", To: "remoteFileName" },
    ]

   Exports: [
        { From: "remoteFileName", To:"/path/localFileName" },
    ]
 }
```

If you want your files to be accessible to multiple jobs then it might make sense to copy them to UNICORE storage and have the jobs access them from there. This can be achieved in the same manner, simply by substituting the local file name with the uri of the file on the UNICORE Target System's storage.

```
{
    Imports: [
       { From: "u6://TS/Storage/fileName", To: "remoteFileName" },
    ]

   Exports: [
        { From: "remoteFileName", To:"u6://TS/Storage/fileName" },
    ]
 }
```

UCC Resources

It is possible to set certain requirements for the resources on which your job will run. For example, you can specify how many nodes or CPUs your job will require.

```
Resources: {
      Memory: 16M,
      CPUs: 32,
      Nodes: 4,
      Runtime: 3600
    }
```

For these exercises we should not set high requirements as we have only limited resources available to us for the tutorial!

UCC localScript.sh

Copy this file to a localScript.sh file:

```
echo "Hello" >> newFile
```

UCC Data Management

The script will need to be available on the worker node in order for it to run. For this exercise you will first copy the script to your home directory on the UNICORE Target System. Your job will then access the script from there.

First copy the script to your remote home directory. The ucc command `put-file` allows you to do this.

```
$ ucc put-file -s localScript.sh -t u6://EMI-UNICOREX/Home/script.sh
```

This command takes a source file with the `-s` option and a target file with the `-t` option.

Now that you have copied the file to your UNICORE home directory, you can view it with the `ucc ls` command.

```
$ ucc ls u6://EMI-UNICOREX/Home
```

You can always copy remote files back to the UI machine with the command `ucc get-file`

```
$ ucc get-file -s u6://EMI-UNICOREX/Home/script.sh -t copyOfLocalScript.sh
```

UCC - bash.u

If you look at the list of applications you will notice that the bash shell is an available application. Now you will submit a simple bash script to run on unicore.

Copy this file to a bash.u file:

```
{
 ApplicationName: "Bash shell",

 Environment: [
                "SOURCE=remoteScript.sh",
        ],

         Imports: [
                { From: "u6://EMI-UNICOREX/Home/script.sh", To: "remoteScript.sh"}
        ],
        Exports: [
                { From: "newFile", To: "localNewFile"},
                { From:"newFile", To: "u6://EMI-UNICOREX/Home/homeNewFile"}
        ],

 Resources: {
        CPUs: 1 ,
 }
}
```

UCC - Running bash job

To run your bash job you just use the same ucc run command:

```
$ ucc run bash.u -v
```

UCC - Passing command line arguments

Now modify the file localScript.sh to take two arguments, for example

```
result=`expr $1 + $2`;
echo "$result";
```

The new script will be called with two numbers as arguments and will return their sum.

To specify the numbers you want to pass as arguments you need to add an "Arguments" line to your job description file

```
Arguments: ["19", "235"],
```

Now try to modify your bash script and bash.u file to take some command line arguments and submit the new job.

UCC - Running on set of files

To run a parameter sweep type job with UNICORE, or any other type of multi-job application, you can use the `ucc batch` command.

First you must create a separate job description file for each job, so for example, if you want to vary the input parameters to your script you would create a job description file for each set of parameters. All of these job description files should be copied to a directory which is passed to the ucc batch command in the `-i` option. The output directory is given with the `-o` option. For example:

```
$ mkdir ex
$ cp *.u ex
$ ucc batch -i ex -o out
```

Note that your job description files will be removed from the input directory, so you should keep a copy of them elsewhere!

Thank you

Thank you for completing the EMI tutorial. Please fill in the feedback form at http://www.surveymonkey.com/s/BW3Z6TV

You are now able to use most of the common commands for compute and data management on an EMI Grid. If you want to test your knowledge with a more complicated and realistic exercise, have a look at the Integrating practical at DCISSIntegratingPractical.

-- KathrynCassidy - 16-Sep-2011

This topic: EMI > EGITF2011UserTutorial
Topic revision: r8 - 2011-09-22 - IvanMartonExCern