

Job submission

Job Submission

The user applications that run in a Grid are called jobs. These are like typical batch jobs on a UNIX/Linux box except that Grid jobs are typically submitted remotely. You can use either the EGRID Ready UI or a system booted from the EGRID Live CD which is configured as a UI. The rest of the chapter provides a basic introduction to jobs, submission of jobs, job monitoring and retrieving the output from a job.

Job Management

The Workload Management System (WMS) (set of services running in the Resource Broker) is responsible for accepting jobs submitted by the user. Then those jobs are dispatched to appropriate CEs² depending on the job requirements and the available resources.

The submission of a job requires GSI authentication between both the user interface and the RB and between RB and the CE. Therefore when submitting, monitoring or retrieving a job output use of a valid proxy certificate is a must. The validity of the certificate should be higher than the execution time of the job.

A special file called the job description file is used to indicate various parameters related to the submitted job. Job description files are written using Job Description Language (JDL). The JDL is used to specify the desired job characteristics and constraints, which are used by the match-making process to select the resources that the job will use.

The WMS retrieves information from the Information System (IS) and the File Catalog and look for the best CE, given the requirement of the submitted job (as given in the job description file). This search process is called "match making".

A wrapper script is automatically created around the actual user job and it is the one that is actually submitted and get executed in a WN. This wrapper performs additional tasks such as correctly setting the job environment and generating logging information. The Logging and Bookkeeping (LB) service running in the RB logs all the job management Grid events. This information can be later used by the users or system administrators for monitoring and troubleshooting.

Job Flow

When a job is submitted to the Grid it has to go through set of steps before completing the execution. Figure 4.1 illustrates the process and set of steps that a successful job has to go through. If the job gets failed and had to be retried the sequence of activities will be slightly different to figure 4.1. Following is an explanation of each of the steps:

- ◆ The job is submitted from the UI to the RB node. The job description file specifies any files needs to be copied from the UI to RB. This set of files is called the Input Sandbox. The event is logged in the LB and status of the job is indicated as *Submitted*.
- ◆ The WMS looks for the best available CE to execute the job based on the Rank and Requirements attributes specified in the JDL file (Section 4.3). For this purpose it queries the BDII for the status of computational and storage resources and the File Catalog to find the location of required data. The event is logged in the LB and status of the job is indicated as *Waiting*.

- ◆ When a suitable match is found the RB prepares the job for submission by creating a wrapper script to be passed, together with other parameters to the selected CE. The event is logged in the LB and status of the job is indicated as *Ready*.
- ◆ The CE receives the request and sends the job for execution to the Local Resource Management System (LRMS). The event is logged in the LB and status of the job is indicated as *Scheduled*.
- ◆ The LRMS handles the job execution on the available local farm of worker nodes. User's files are copied from the RB to the WN where the job is executed. The event is logged in the LB and the status of the job is *Running*.
- ◆ While the job runs, Grid files can be accessed on a close SE using either the RFIO or (Remote File Input/Output), GridFTP or gsiftp or gsidcap protocols, or from remote SEs after copying those locally to the WN local file system with the data management tools.

The job can produce new output data that can be uploaded to the Grid and made available for other Grid users to use (in the current implementation this needs to be done manually). This can be achieved using the Data Management tools described later. Uploading a file to the Grid means copying it on to a SE and registering it in the file catalog. At the same time, during job execution or from the UI, data files can be replicated between two SEs using again the Data Management tools.

- ◆ If the job reaches the end without errors, the output (not large data files, but just small output files specified by the user in the so called Output Sandbox) is transferred back to the RB node. The event is logged in the LB and the status of the job is *Done*.
- ◆ At this point, the user can retrieve the output of his/her job from the UI. The event is logged in the LB and the status of the job is *Cleared*.

Queries of the job status are addressed to the LB database from the UI machine. In addition, from the UI it is possible to query the BDII for status of the resources.

If the CE where the job is assigned is unable to accept/execute it, the job will be automatically resubmitted to another CE that still satisfies the user requirements. When the maximum allowed number of submissions is reached and if still the job is unable to complete the job will be marked as *Aborted*. Users can get information about what happened by querying the LB service.

Table 4.1 summarises the meaning of each the states.

Table 4.1 – Definition of the different job states

Submitted	The job has been submitted by the user but not yet processed by the Network Server
Waiting	The job has been accepted by the Network Server but not yet processed by the Workload Manager
Ready	The job has been assigned to a Computing Element but not yet transferred to it
Scheduled	The job is waiting in the Computing Element's queue
Running	The job is running
Done	The job has finished
Aborted	The job has been aborted by the WMS (e.g. because it was too long, or the proxy certificated expired, etc.)
Canceled	The job has been canceled by the user
Cleared	The Output Sandbox has been transferred to the User Interface

The Job Description Language

A special file called the job description file (.jdl file) is used to describe the job being submitted. It indicates various parameters related to the job. It is written using the Job Description Language (JDL). The JDL can be used to indicate any input files to be transferred from the UI to the RB, output files to be produced, specific resource requirements such as the number of CPUs, disk space, presence of specific runtime libraries in WNs, presence of the specified file in close SE, etc.

Only a brief introduction to JDL is given here more detailed information can be found in [4.1].

Figure 4.1 – Successful job flow on EGEE/LCG Grid

A JDL file consists of sequence of statements having the following format:

```
attribute = value;
```

Literal strings (for values) should be enclosed in double quotes ("). If a string itself contains double quotes, they must be escaped with a backslash (e.g.: `Arguments = "\"hello\"`). Do not use characters `"`, `&`, `<`, etc in the `Arguments` attribute of the JDL; if those characters are to be used, they must be written in escaped form: `\\`, `\&`, `\<`.

Comments must be preceded by a sharp character (#) or have to follow the C++ syntax, i.e a double slash (//) at the beginning of each line or statements begun/ended respectively with `/*` and `*/`.

Warning

Warning:

The JDL tends to be sensitive to blank characters and tabs. No blank characters or tabs should follow the semicolon at the end of a line.

In a job description file, some attributes are mandatory, while some others are optional. At a minimum, the user must at least specify the name of the executable, the files where to write the standard output and the standard error of the job (they can even be the same file). Consider the following example:

```
Executable = "/bin/echo";  
StdOutput = "std.out";  
StdError = "std.err";
```

In the above example, the job should execute the echo utility and all the outputs of the Shell script should be saved in the `std.out` file. If there are any errors, those will be saved in the `std.err` file.

Note

Note:

`Executable` is the only mandatory attribute in a JDL.

If any arguments are needed to be passed to the executable program it can be done with the use of `Arguments` attribute. Following example illustrates the use of `Arguments` attribute for the above created JDL:

```
Executable = "/bin/echo";
```

```
Arguments = "Hello World";
StdOutput = "std.out";
StdError = "std.err";
```

If any inputs are needed to be given to the executable through the standard input, an input file can be specified with the `StdInput2` attribute as follows:

```
StdInput = "std.in";
```

Then, the files to be transferred between the UI and the WN before and after the job execution can be specified. The files that need to be transferred before the execution begins are called the *Input Sandbox* while the file containing results of the job is called the *Output Sandbox*. Following example illustrates the use of `InputSandbox2` and `OutputSandbox2` attributes:

```
InputSandbox = {"std.in"};
OutputSandbox = {"std.out", "std.err"};
```

Note

Note:

Using an *Output Sandbox* is not the only way of retrieving (i.e. copying) files from the Grid. Instead data management commands can be used (section 5). If the output files are very large, it is advisable not to retrieve them as part of the Output Sandbox.

If the executable does not reside on the WN it should also be passed to the WN as part of the Input Sandbox as in the following example:

```
InputSandbox = {"test.sh", "std.in"};
```

Wildcards are allowed only in the `InputSandbox2` attribute. The list of files in the *Input Sandbox* is specified relatively to the current working directory (it is the local Linux/UNIX directory where the job submission command is invoked). Absolute paths cannot be specified in the `OutputSandbox2` attribute. Neither the `InputSandbox2` nor the `OutputSandbox2` lists can contain two files with the same name (even if in different paths) as when transferred, they would overwrite each other.

If the submitted job depends on any environment variables the environment of the job can be modified using the `Environment` attribute. For example:

```
Environment = {"CMS_PATH=$HOME/cms", "CMS_DB=$CMS_PATH/cmdb"};
```

Imposing Constraints on the CE

The `Requirements` attribute can be used to express any kind of constraint on the resources where the job can run. Its value is a Boolean expression that must evaluate to *True* for a job to run on that specific CE. For this purpose all the GLUE attributes of the IS can be used.

Note

Note:

Only one `Requirements` attribute can be specified (if there is more than one, only the last one is considered). If several conditions must be applied to the job, then they all must be included in a single `Requirements` attribute, using a Boolean expression.

For an example let us suppose that the job needs to be run on a CE, whose `WNS2` have at least two `CPUs2`. Then the `Requirements` attribute should have the following value:

```
Requirements = other.GlueCEInfoTotalCPUs > 1;
```

where the `other .` prefix is used to indicate that the `GlueCEInfoLRMSType2` attribute refers to the CE characteristics and not to those of the job. If `other .` prefix is not specified, then the default `self .` prefix is assumed, indicating that the attribute refers to the job characteristics description.

So, as a general rule, requirements on attributes of a CE are written prefixing `other .` to the attribute name in the IS schema.

The Grid user can also ask the RB to send a job to a particular CE with the following expression:

```
Requirements = other.GlueCEUniqueID == "ce-01.egrid.it:2119/jobmanager-pbs-short";
```

If the job must run on a CE where particular VO software is installed and this information is published by the CE as a software tag, something like the following must be written:

```
Requirements = Member("VO-egrid-client", other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

Note

Note:

The `Member` operator is used to test if its first argument is a member of its second argument (a list). In this example, the `GlueHostApplicationSoftwareRunTimeEnvironment2` attribute is a list.

The `VirtualOrganisation2` attribute represents another way to specify the VO of the user, as in:

```
VirtualOrganisation = "egrid";
```

The JDL attribute called `RetryCount2` can be used to specify how many times the RB must try to resubmit a job if it fails due to some error which is not the job itself.

The `MyProxyServer2` attribute indicates the Proxy Server containing the user's long-term proxy that the WMS must use to renew the proxy certificate when it is about to expire. If this attribute is not included manually by the user, then it is automatically added when the job is submitted.

Submitting Your First Job

Let us try to submit a simple job to the Grid.

Step 1

First of all you need to write the user application that needs to be executed on the Grid. In this case we will use the `echo` utility which is available in any UNIX/Linux host. Therefore you do not need to develop any application.

Step 2

Next step is to create the corresponding JDL file. Use your favourite text editor and create a file called `Hello.jdl`:

```
$ vi Hello.jdl
```

Add the following set of lines to the `Hello.jdl` file:

```
Executable = "/bin/echo";
Arguments = "Hello World!";
StdOutput = "std.out";
StdError = "std.err";
OutputSandbox = {"stdout", "stderr"};
```

The above `.jdl` will execute the `echo` utility on the WN and it will print `Hello World!`. However the output is redirected to the `std.out` file. Similarly if there are any errors those will be directed to the `std.err` file.

Step 3

Before submitting any jobs for the first time you need to create a valid proxy certificate. Use the `grid-proxy-init` command to create a certificate as in the following example:

```
$ grid-proxy-init
Your identity:
/O=GRID@Trieste/DC=it/DC=sisssa/DC=grid/OU=people/CN=Dilum Bandara
Enter GRID pass phrase for this identity:
Creating proxy ..... Done
Your proxy is valid until: Sat Jul 22 05:15:57 2006
```

Step 4

For submission we use the `edg-job-submit` command which has the following format:

```
edg-job-submit [options]_? "jdl file"
```

Where *[options]_?* defines various options accepted by command and *"jdl file"* defines the path to the `.jdl` file that is to be submitted.

All options supported by the `edg-job-submit` command can be found by using the `--help` option as follows:

```
$ edg-job-submit -help
```

Step 5

Let us try to submit our job to `gridats`. Use the following command:

```
$ edg-job-submit --vo gridats Hello.jdl
```

The output of a successful submission is similar to:

```
Selected Virtual Organisation name (from --vo option): gridats
Connecting to host gridts05.grid.elettra.trieste.it, port 7772
Logging to host gridts05.grid.elettra.trieste.it, port 9002
```

```
*****
                                JOB SUBMIT OUTCOME
The job has been successfully submitted to the Network Server.
Use edg-job-status command to check job current status. Your job identifier (edg_jobId) is
- https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jReEPvRJQ
*****
```

The command returns to the user the job identifier (`edg_jobId`), which defines uniquely the job and

can be used to perform further operations on the job, like querying the system about its status, or canceling it. The format of the job Id is:

[https://LBserver_address\[:port\]/unique_string](https://LBserver_address[:port]/unique_string)

where *unique_string* is guaranteed to be unique and *LBserver?_address* is the address of the Logging and Bookkeeping server for the job, which usually (but not necessarily) is also the RB.

Note

Note:

Although this looks like a URL it does not identify a web page.

If the command returns the following error:

```
**** Error: API_NATIVE_ERROR ****
Error while calling the "NSClient::multi" native api
AuthenticationException: Failed to establish security context...
**** Error: UI_NO_NS_CONTACT ****
Unable to contact any Network Server
```

it means that there are authentication problems between the UI and the network server (check your proxy certificate or have the site administrator check the certificate of the server). This may also happen to do to the incorrect time and timezone settings of the host.

Monitoring Job Execution

After a job has been successfully submitted its status and its history can be retrieved form time to time using the `edg-job-status` command. The command has the following format:

```
edg-job-status [options]? <job Id> [<other job ids> ...]?
```

Step 1

Lets us try to find out the status of the job which was submitted earlier. To find out information about a job you need its `jobId`. Following example illustrates the use of `edg-job-status` command:

```
$ edg-job-status https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jReEPvRJQ
```

(Make sure you replace the above jobId with the correct one.) The output of the command will be similar to the following:

```
*****
BOOKKEEPING INFORMATION:

Status info for the Job : https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jReEP
Current Status: Scheduled
Status Reason: Job successfully submitted to Globus
Destination: ce-01.grid.sissa.it:2119/jobmanager-lcgpbs-gridats
reached on: Fri Jul 21 15:17:05 2006
```

The above output explains the following:

Current Status:

indicates the status of the job submitted.

Status Reason:

indicates any reasons for the above given status.

Destination:

indicates in which CE your job is scheduled to run

reached on:

label indicates the time and date that the job entered the given status.

To find out list of options supported by the `edg-job-status` command use the `--help` option.

```
$ edg-job-status --help
```

It will give you a detailed description about the command and list of available options.

Step 2

From time to time issue the above command and see how the job status changes.

The `watch` command may be useful in monitoring a job status:

```
$ watch -n 30 edg-job-status https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jR
```

Exit the `watch` session by pressing `Ctrl+C`.

Step 4

You can use the `--verbosity` option to find out more information about the submitted job. Verbose level can be 0, 1 or 2.

Verbosity 0 is the default verbosity level.

Let us try with a verbosity of 1:

```
$ edg-job-status --verbosity 1 https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2
```

BOOKKEEPING INFORMATION:

Status info for the Job : https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jReEP

Current Status: Done (Success)

Exit code: 0

Status Reason: Job terminated successfully

Destination: grid001.oat.ts.astro.it:2119/jobmanager-lcgpbs-grid

reached on: Fri Jul 21 15:30:16 2006

- cancelling = 0

- ce_node = gridts01.grid.elettra.trieste.it

- children_hist = 0

- children_num = 0

- condorId = 31087

- cpuTime = 0

- destination = gridts01.grid.elettra.trieste.it:2119/jobmanager-lcgpbs-gridats

- done_code = 0

- expectUpdate = 0

```

- jobtype = 0
- lastUpdateTime = Fri Jul 21 15:26:45 2006
- location = none
- network_server = gridts05.grid.elettra.trieste.it:7772
- owner = /O=GRID@Trieste/DC=it/DC=sissa/DC=grid/OU=people/CN=Dilum Bandara
- resubmitted = 0
- seed = uLU0BArrdV98041PLThJ5Q
- subjob_failed = 0

```

This time you will see more detailed information about the job. Such details will be useful when trying to locate problems.

Now try with a verbosity of 2:

```
$ edg-job-status --verbosity 2 https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2
```

This time lots of information about the job (too much to be displayed here!) will appear. Other than for debugging purposes such details are not needed for the user.

Retrieving an Output of a Job

If the job was successfully completed (it reached the Done status), the output files defined using the `OutputSandbox_2` attribute can be retrieved to the UI. For this purpose, the `edg-job-get-output` command is used. The `edg-job-get-output` command has the following format:

```
edg-job-get-output [options]_2 <job id> [<other job id> ...]_2
```

More information about the command can be retrieved using the `--help` option.

```
$ edg-job-get-output --help
```

Step 1

For instance, let us try to retrieve the output of the previous job. Use the following command (make sure to put the correct `jobId`):

```
$ edg-job-get-output https://gridts05.grid.elettra.trieste.it:9000/k9fyFw23xnkL2jReEPvRJQ
```

The output will be similar to the following:

```
Retrieving files from host: gridts05.grid.elettra.trieste.it ( for https://gridts05.grid.e
```

```
JOB GET OUTPUT OUTCOME
```

```
Output sandbox files for the job:
```

```
- https://gridts05.grid.elettra.trieste.it:9000/JFaPdjjjsjx8NYjMVfhIpA
```

```
have been successfully retrieved and stored in the directory:
```

```
/usr/egrid-ready-ui/joboutput/dilumb_JFaPdjjjsjx8NYjMVfhIpA
```

The above command will retrieve the output from RB and save it in a new directory (the name of the directory is derived from the *unique_string* which is part of the job Id). The absolute path to the newly created directory is given at the end of the results retrieved by the `edg-job-get-output` command. For the above example, the output directory is:

```
/usr/egrid-ready-ui/joboutput/dilumb_JFaPdjjjsjx8NYjMVfhIpA.
```

Step 2

Let us try to see the results of the job. First you need to go to the directory where your outputs are saved:

```
$ cd /usr/egrid-ready-ui/joboutput/dilumb_JFaPdjjjsjx8NYjMVfhIpA
```

Then list the contents of that directory:

```
$ ls
stderr stdout
```

The output of the job will be saved in the `std.out` file while any errors will be saved in the `std.err` file.

Use the `more` or `cat` command to see the content of each file:

```
$ cat std.out
Hello World!
$ cat std.err
```

The `std.err` file is empty, since no errors occurred during the execution of the job.

Listing CEs? that match a Job Description

It is possible to see which CEs? are eligible to run a job specified by a given JDL file using the command `edg-job-list-match`. It checks whether necessary resources to run the job are available and whether the given JDL file is rational.

Step 1

Let us try to see what CEs? are capable of executing our `Hello.jdl`:

```
$ edg-job-list-match --vo gridats Hello.jdl
```

The output will be similar to the following:

```
Selected Virtual Organisation name: gridats
```

```
Connecting to host gridts05.grid.elettra.trieste.it, port 7772
```

```
*****
```

```
COMPUTING ELEMENT IDs LIST
```

The following CE(s) matching your job requirements have been found:

```
*CEId*
```

```
baciuco.grid.sissa.it:2119/jobmanager-pbs-gridats
ce-01.grid.sissa.it:2119/jobmanager-lcgpbs-gridats
egrid-ce-01.egrid.it:2119/jobmanager-lcgpbs-gridats
egrid.create-net.org:2119/jobmanager-lcgpbs-gridats
```

```
grid001.oat.ts.astro.it:2119/jobmanager-lcgpbs-grid
gridts01.grid.elettra.trieste.it:2119/jobmanager-lcgpbs-gridats
```

```
*****
```

It gives us a list of CEs₂ where the given job can be run. This information would be important when deciding which are the local CEs₂, closeSEs, etc.

Saving Job IDs₂ to a File

Instead of typing or copying the jobId all the time you can save it in a file and those saved jobIds can be used later with commands such as `edg-job-status` and `edg-job-get-output`.

To save the jobId to a separate file use the `--output` option as in the following example:

```
$ edg-job-submit --vo gridats --output jobIDs Hello.jdl
```

```
Selected Virtual Organisation name (from --vo option): gridats
Connecting to host gridts05.grid.elettra.trieste.it, port 7772
Logging to host gridts05.grid.elettra.trieste.it, port 9002
```

```
===== edg-job-submit Success =====
The job has been successfully submitted to the Network Server.
Use edg-job-status command to check job current status. Your job identifier (edg_jobId) is:

- https://gridts05.grid.elettra.trieste.it:9000/fdXICj5xICsf571bmEHw0Q

The edg_jobId has been saved in the following file:
/home/dilumb/jobIDs
=====
```

Above command display the jobId, at the same time the jobId is also saved in the `jobIDs` file. If the file already exists it will append the new jobId to it.

Let us try to find out the status of the job that was just submitted without manually giving the jobId. Use the following command; the jobId is passed to the command using the `--input` option:

```
$ edg-job-status --input jobIDs
```

Now jobId will be taken automatically from the `jobIDs` file. The output will show no difference from invocations of `edg-job-status` where the Job Id is given on the command line.

To see how multiple job Ids are handled, submit another instance of the same `Hello.jdl`:

```
$ edg-job-submit --vo gridats --output jobIDs Hello.jdl
```

The new job Id will be appended to the already existing `jobIDs` file.

Now retrieve the status of the job that was just submitted. Use the following command:

```
$ edg-job-status --input jobIDs
```

Since there are 2 jobIds in the `jobIDs` file, the command will ask you to select one or more jobs to display. Type the number given in front of the jobId to display the job status and press *Enter*. If you want to display the status of all jobs type `a` and press *Enter*. If no jobs to be displayed type `q` and press *Enter*. See the following example:

```
$ edg-job-status --input jobIDs
```

```
-----  
1 : https://gridts05.grid.elettra.trieste.it:9000/fdXICj5xICsf571bmEHw0Q  
2 : https://gridts05.grid.elettra.trieste.it:9000/fdXICj5xICsf571bmEHwZZ  
a : all  
q : quit  
-----
```

Choose one or more `edg_jobId(s)` in the list - [1-2]all:

After you have chosen a job Id or all, the statuses of the selected jobs will be output as if you had typed the job Ids on the command line of `edg-job-status`.

Submitting Jobs to a Specific Computing Element

The `-r` option can be used to submit jobs to a specific CE. However in this case no match making process will take effect. It is completely the responsibility of the user to make sure the selected CE has all the necessary resource for successful execution of the job.

Step 1

Let us try to submit our `Hello.jdl` to CE `baciuco.grid.sissa.it`.

When we are using the `-r` option we need to give the queue ID of the CE. The ID of the CE can be found either using `lcg-infosites`, `lcg-info` or `edg-job-list-match` command.

To submit our job to Baciuco use the following command:

```
$ edg-job-submit \  
-r baciuco.grid.sissa.it:2119/jobmanager-pbs-gridats \  
--output jobIDs Hello.jdl
```

A possible output would be:

```
Selected Virtual Organisation name (from UI conf file): gridats  
Connecting to host gridts05.grid.elettra.trieste.it, port 7772  
Logging to host gridts05.grid.elettra.trieste.it, port 9002
```

```
***** edg-job-submit Success *****
```

The job has been successfully submitted to the Network Server.

Use `edg-job-status` command to check job current status. Your job identifier (`edg_jobId`) is:

```
- https://gridts05.grid.elettra.trieste.it:9000/z4jNzENxQ84_xTg3-lNkIw
```

The `edg_jobId` has been saved in the following file:

```
/home/dilumb/jobIDs
```

```
*****
```

Step 2

Then you can use the `edg-job-status` command to monitor the status of job as earlier.

```
$ edg-job-status --input  
jobIDs
```

Step 3

When the job is finished (when the status is *Done*) retrieve the output using the following command:

```
$ edg-job-get-output --input jobIDs
```

Cancelling a Job

Any job that has been submitted can be cancelled, using the `edg-job-cancel` command which has the following format:

```
edg-job-cancel [options]? <job id> [<other job ids> ...]?
```

To find out more details about the command use the `--help` option:

```
$ edg-job-cancel -help
```

For instance, let us cancel the job that was submitted one before last. Use the following command to cancel the job (make sure to substitute with the correct job Id):

```
$ edg-job-cancel https://gridts05.grid.elettra.trieste.it:9000/ZexpWslqoVSnk_IQmqV92w
Are you sure you want to remove specified job(s)? [y/n]n :y
```

```
***** edg-job-cancel Success *****
```

The cancellation request has been successfully submitted for the following job(s):

```
- https://gridts05.grid.elettra.trieste.it:9000/ZexpWslqoVSnk_IQmqV92w
```

```
*****
```

You can also use the `--input` option to provide the job Id as in the following example:

```
$ edg-job-cancel --input jobIDs
```

Retrieving Logging Information About Submitted Jobs

You can retrieve job logging information for jobs previously submitted with the `edg-job-submit` command from the LB database. This information can only be retrieved when the job has finished its life cycle. This feature is specially useful in analysing job failures. This information can be retrieved using the `edg-job-get-logging-info` command which has the following format:

```
edg-job-get-logging-info [options]? <job id> [<other job ids> ...]?
```

For example, let us try to retrieve logging information for the 1st job that was submitted (make sure to use the correct job Id). Output of the command can be really lengthy depending on the verbosity level that is selected. Therefore it would be a good idea to save the output to a file using the `--output` option. Try the following command with correct jobId:

```
$ edg-job-get-logging-info \
  --verbosity 2 \
  --output logging_file.txt \
  https://gridts05.grid.elettra.trieste.it:9000/JFaPdjjjsjx8NYjMVfhIpA
```

```
***** edg-job-get-logging-info Success *****
```

Logging Information has been found and stored in the file:

```
/home/dilumb/logging_file.txt
```

```
*****
```

The output will be saved into the `logging_file.txt` file in the current directory. To see the content of the file use the following command:

```
$ less logging_file.txt
```

The level of job information being listed can be controlled by varying the verbosity level.

More on Jobs

The job that we discussed so far is really simple. The actual jobs submitted on a Grid can be really complex. Let us try few more jobs which illustrate use of job submission, monitoring and retrieval features.

Step 1

This time we will be sending a Shell script from the UI to the CE for execution. Create the following Shell script with a suitable text editor then save it as `info.sh`:

```
#!/bin/bash

#where it is running
hostname > hostname.txt

#Type of CPU
cp /proc/cpuinfo cpuinfo.txt

#working directory
pwd > working_dir.txt

#archive and compress all the files
tar cvjf results.tar.bz2 \
    hostname.txt cpuinfo.txt working_dir.txt
```

The above Shell script tries to collect some information about the WN that it gets executed in to several files. Then at the end those files are archived together.

Step 2

Create the `info.jdl` file with the following attributes and values:

```
Executable = "info.sh";
StdOutput = "std.out";
StdError = "std.err";
InputSandbox = {"info.sh"};
OutputSandbox = {"results.tar.bz2", "std.out", "std.err"};
```

Step 3

Double check both the `info.sh` and `nfo.jdl` files, then use the following command to submit the job:

```
$ edg-job-submit -vo gridats --output jobIDs info.jdl
```

If the job submission is successful the new `jobId` will be saved in the `jobIDs` file.

Step 4

Use the `edg-job-status` command to check the status of the job that was just submitted. Occasionally check whether the job has finished without any errors (i.e. has reached the *Done* state).

Use the following command to check the job status:

```
$ edg-job-status --input jobIDs
```

Step 5

When the job is successfully completed retrieve the output using `edg-job-get-output` command as follows:

```
$ edg-job-get-output --input jobIDs
```

Then the files given in the Output Sandbox will be copied to the UI. You should get the `results.tar.bz2`, `std.out` and `std.err` files.

Step 6

Go to the directory where above files are stored. Then check the content of both `std.out`, `std.err` files. Use the following command:

```
$ less std.out std.err
```

Step 7

To extract the compressed archive `results.tar.bz2` use the following command:

```
$ tar xvfj results.tar.bz2
```

Then `hostname.txt`, `cpuinfo.txt` and `working_dir.txt` file will appear.

Step 8

Check the content of each file; you will get an output similar to the following:

```
$ cat hostname.txt
wn-01.grid.trieste.it
```

```
$ cat cpuinfo.txt
processor      : 0
vendor_id    : GenuineIntel
cpu family   : 15
model        : 2
model name   : Intel(R) Pentium(R) 4 CPU 2.60GHz
stepping     : 9
cpu MHz      : 2593.654
cache size   : 512 KB
[.....]
```

```
$ cat working_dir.txt
total: used: free: shared: buffers: cached:
Mem: 1049833472 988823552 61009920 0 128397312 741892096
Swap: 1073733632 0 1073733632
MemTotal: 1025228 kB
MemFree: 59580 kB
MemShared: 0 kB
Buffers: 125388 kB
Cached: 724504 kB
SwapCached: 0 kB
Active: 301532 kB
[.....]
```

Jobs that can run in Parallel

Some jobs can be divided into multiple jobs and can be executed on different WNs² at the same time. To divide a job into multiple sub jobs there should not be any dependence in job elements. Suppose that we need to find the value of first 1000 elements of the following function:

This can be done as two different jobs where the first job try to calculate y for the x in $[1, 500]$ ² while the second job try to calculate y for the x in $[501, 1000]$ ². Now the workload is divided among two jobs therefore execution time of the job should be 1/2 of the initial value. If the above range of x is divided into 10 equal ranges and submitted as 10 different jobs then the total execution time should be 1/10 of the initial value.

Let us write a simple Shell script to illustrate execution of the above equation.

Step 1

Create the following Shell script and save it as computeF.sh:

```
#!/bin/bash
y=0

for( (x=$1;x<$2;x+=3) )
do

let y=$x*5+1
echo "X: $x Y: $y"

#make the algorithm run slowly
sleep 1

done
```

The above Shell script accepts three inputs (\$1, \$2, \$3) and those are used to control the for loop. The 1st input defines the starting value of the loop, 2nd input defines the ending value of the loop and 3rd input defines the increment. The sleep system call is used to slow the execution of the algorithm so that we have enough time to monitor its execution.

Step 2

Let us submit the above Shell script as two different jobs, 1st one having value of x in $[1, 500]$ ² and the 2nd one having value of x in $[501, 1000]$ ².

Create the computeF1.jdl file and add the following set of lines:

```
Executable = "computeF.sh";
Arguments = "0 500 1";
StdOutput = "std.out";
StdError="std.err";
InputSandbox = {"computeF.sh"};
OutputSandbox = {"std.out", "std.err"};
```

Create the computeF2.jdl file and add the following set of lines:

```
Executable = "computeF.sh";
Arguments = "501 1000 1";
StdOutput = "std.out";
StdError="std.err";
InputSandbox = {"computeF.sh"};
OutputSandbox = {"std.out", "std.err"};
```

The only difference between `computeF1.jdl` and `computeF2.jdl` is in the `Arguments` attribute.

Step 3

Make sure all the files are there and as well as no syntax mistakes. Submit each `.jdl` using the following commands:

```
$ edg-job-submit --vo gridats --output jobIDs computeF1.jdl
$ edg-job-submit --vo gridats --output jobIDs computeF2.jdl
```

Step 4

Monitor the execution of each job. When each job is finished retrieve the output. The output will be in the `std.out` file.

Scripting for More Advanced Job Submission

Job submission, monitoring and retrieving output can be automated using shell scripts. In the example that was discussed in section 4.10.1 we had to manually submit the job twice. This may be ok since a couple of jobs. If there is large number of jobs to be submitted at the same time it would be a good idea to automate job submission. Following example illustrates how we can automate the job submission in section 4.10.1.

Step 1

Copy `computeF.sh`, `computeF1.jdl` and `computeF2.jdl` files to a separate directory:

```
$ mkdir ./tmpJobs
$ cp computeF.sh ./tmpJobs
$ cp computeF1.jdl ./tmpJobs
$ cp computeF2.jdl ./tmpJobs
$ cd ./tmpJobs
$ ls
computeF.sh computeF1.jdl computeF2.jdl
```

Step 2

Create a new Shell script with the following set of lines and save it as `JobSubmit?.sh`:

```
#!/bin/bash

# Submit all the .jdl files in the directory to gridats
for i in *.jdl;
do
    edg-job-submit --vo gridats --output jobIDs $i
done
```

Step 3

Set the execution privileges to the Shell script. Use the following command:

```
$ chmod +x JobSubmit.sh
```

Step 4

Then execute the Shell script to submit all the jobs in the directory as follows:

```
$ ./JobSubmit.sh
```

The output will of course be the concatenated output of the JDL submission commands; job Ids will be saved in the `jobIDs` file.

Step 4

Monitor the execution of each job using `edg-job-status` command.

When each job is finished retrieve the output. The output will be saved in the individual `std.out` files.

Shell scripting can be used to even generate `.jdl` files depending on some user inputs. If you want to submit multiple instances of the `computeF.sh` script with different input parameters you do not need to manually generate `computeF1.jdl`, `computeF2.jdl`, ..., `computeFn.jdl` files. A carefully written Shell script can even generate individual `.jdl` files and another Shell script like the previous one can be used to submit all those jobs automatically one after another.

How much you can automate job submission, monitoring and job output retrieval depends only on how good at you are in writing complex shell scripts. So try a bit more with various Shell scripts.